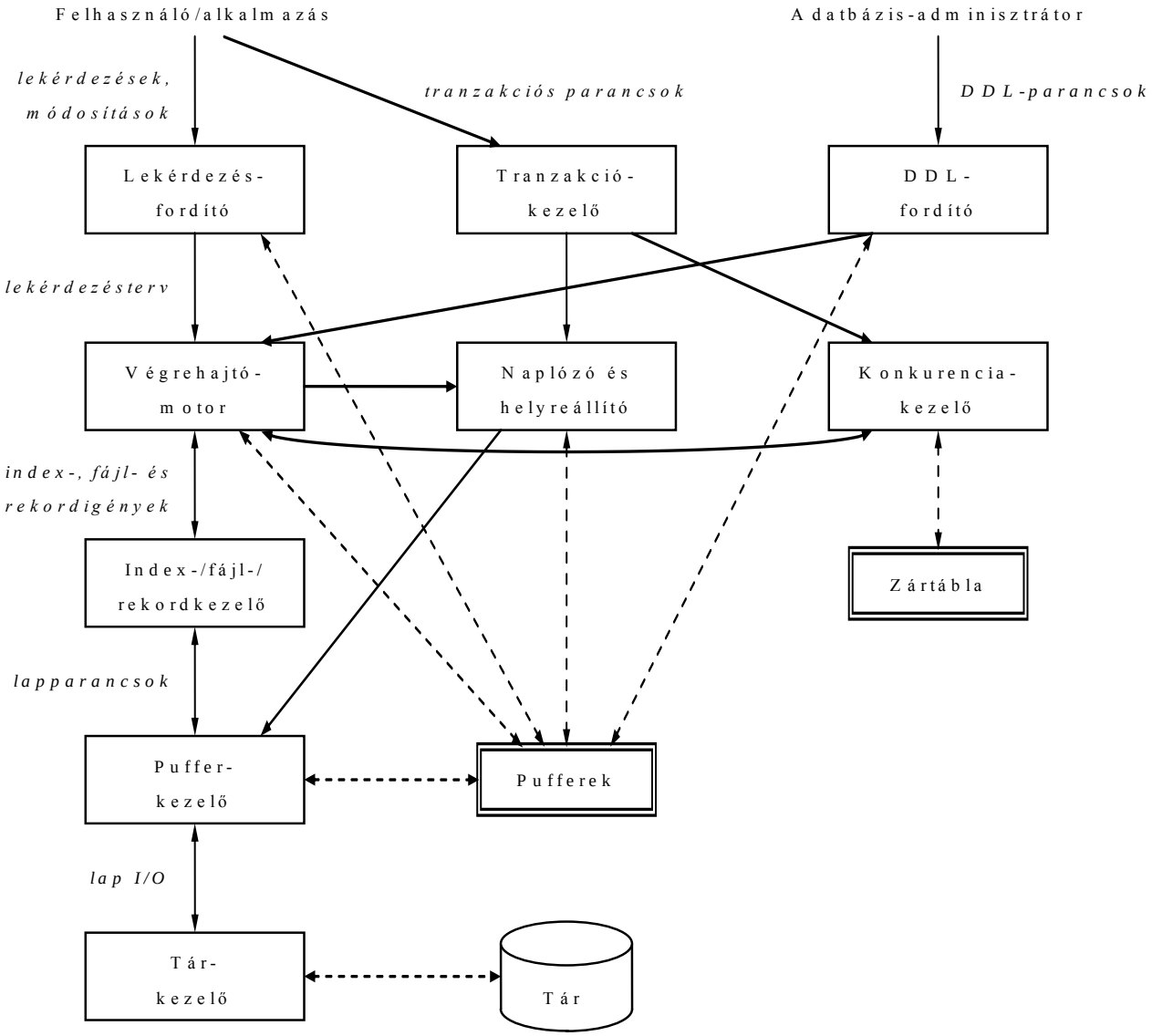
Adatbázisrendszerek megvalósítása 2

Irodalom: Hector Garcia-Molina – Jeffrey D. Ullman – Jennifer Widom: Adatbázisrendszerek megvalósítása, 6. és 7. fejezet

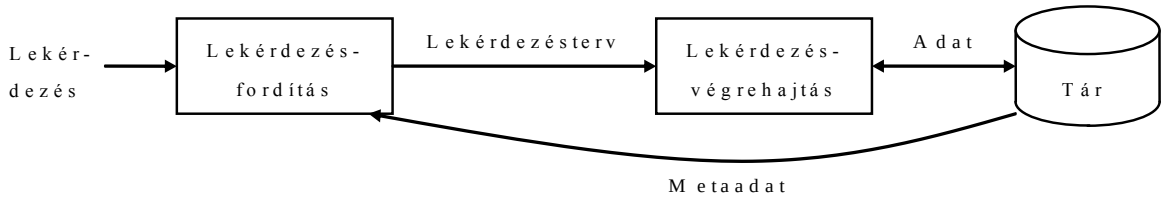
Előfeltételek: Adatbázisrendszerek tárgy, SQL.

Tartalom: A lekérdezésfordító.

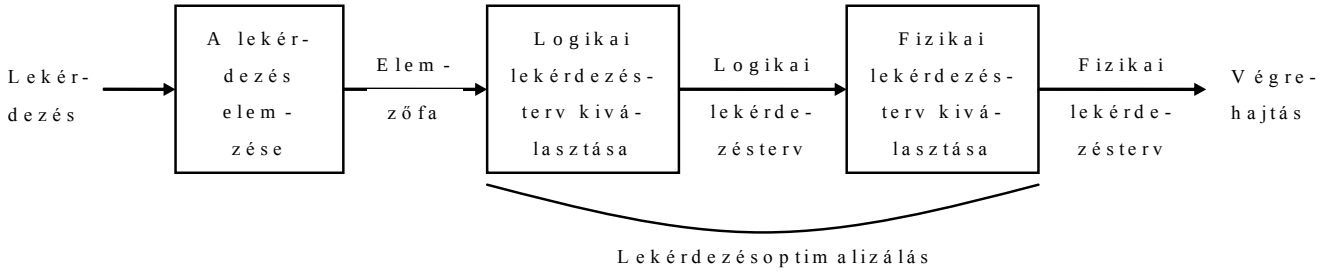
A lekérdezésfordító helye az adatbázis-kezelő rendszerben



A *lekérdezésfeldolgozó* (query processor) két fő részből áll: a lekérdezésfordítóból, amely a lekérdezésből előállít egy lekérdezéstervet, valamint a lekérdezés-végrehajtóból, amely elvégzi a lekérdezéstervben szereplő műveleteket:



A lekérdezésfordítás menete a következő:



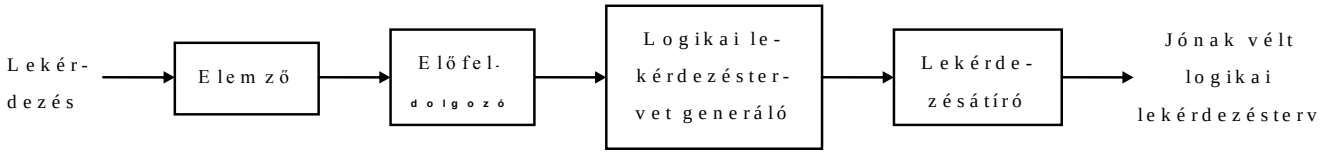
A lekérdezésfordító feladatát az alábbi három fő lépésben végzi el:

1. Az SQL nyelven megfogalmazott lekérdezés (mint sztring) *elemzése* (parsing), azaz átalakítása egy olyan kifejezésfává, amely a lekérdezés szerkezetének egy jól használható reprezentációját adja. Ezt a kifejezésfát *elemzőfának* (parse tree) nevezzük. Az elemzőfa levélelemei a lekérdezés tokenei.
2. Az elemzőfa átalakítása egy relációalgebrával (vagy más hasonló absztrakt lekérdezőnyelv jelölésrendszerével) megfogalmazott kifejezésfává, amit *logikai lekérdezéstervnek* (logical query plan) nevezünk. Ez a *lekérdezésátírás* (query rewrite). A logikai lekérdezésterv levélelemei relációk, a közbülső elemei pedig relációalgebrai operátorok.
3. A logikai lekérdezésterv átalakítása egy olyan *fizikai lekérdezéstervvé* (physical query plan), amely már nem csak a végrehajtásra kerülő műveleteket mutatja, hanem ezek végrehajtási sorrendjét, az egyes műveletek végrehajtásához használt algoritmusokat is, továbbá azt is, hogy a tárolt adatokat hogyan kapjuk meg (például kell-e rendezni valamelyik relációt, és ha igen, mikor), és hogy az adatokat hogyan adják át egymásnak a műveletek (csővezetéken keresztül, memóriapufferekben vagy lemezen). A fizikai tervet is egy kifejezésfával ábrázoljuk.

Az elemzés eredménye tehát egy elemzőfa. Ebből előállítunk egy kiindulási logikai lekérdezéstervet, amelyből különböző relációalgebrai szabályok segítségével javított változatokat készítünk, amelyek ekvivalensek az eredetivel, de hatékonyabbak. Amikor egy logikai tervből fizikai lekérdezéstervet készítünk, fel kell mérnünk az egyes választási lehetőségek várható költségét. A költségbecslés maga is egy önálló tudomány. A 2. és 3. lépéseket együtt *lekérdezésoptimalizálásnak* (query optimization) nevezzük. Ennek folyamán valamennyi döntés az adatbázis metaadataitól függ, mint például az egyes relációk mérete, egy attribútum különböző értékeinek gyakorisága vagy bizonyos indexek létezése.

Elemzés

A lekérdezések fordításának első fázisait a következő ábra illusztrálja:



Az ábrán látható négy doboz az előző ábra első két lépésének felel meg.

Szintaktikus elemzés és elemzőfák

Az elemző feladata, hogy egy SQL-ben vagy ahhoz hasonló nyelven megírt szöveget egy elemzőfává konvertáljon. Az *elemzőfa* (parse tree) csomópontjai az alábbiak lehetnek:

- *Atomok* (terminálisok), melyek lexikai elemek, mint például kulcsszavak, attribútumok vagy relációk nevei, konstansok, zárójelek, operátorok és egyéb sémaelemek. Az atomokat jelképező csomópontoknak nincsenek gyerekei.
- *Szintaktikus kategóriák* (nemterminálisok), melyek olyan nevek, amelyek a lekérdezések hasonló szerepet betöltő részegységeit képviselik. A szintaktikus kategóriákat <> jelek közé tett informatív nevekkel jelöljük. Az `<SFW>` például a megszokott `SELECT…FROM…WHERE` alakú lekérdezéseket reprezentálja, míg a `<Feltétel>` egy tetszőleges logikai kifejezést jelöl. A szintaktikus kategóriákat képviselő csomópontoknak a gyerekeit a nyelvet megadó nyelvtan valamely szabálya írja le. Annak részletei, hogy egy nyelvet definiáló nyelvtant hogyan lehet megtervezni, illetve hogy az elemzés maga (a lekérdezés elemzőfává történő alakítása) hogyan történik, más kurzusok témája (Nyelvek és automaták, Fordítóprogramok).

Egy leegyszerűsített SQL nyelvet leíró nyelvtan

Az elemzési folyamat bemutatásához megadunk néhány szabályt, amelyekkel az SQL egy részhalmazát adó lekérdezőnyelvet definiálunk. Kitérünk majd arra is, hogy milyen további szabályok kellenének ahhoz, hogy a teljes SQL-t leíró nyelvtant alkossunk.

```
<Lekérdezés> ::= <SFW>
<Lekérdezés> ::= ( <Lekérdezés> )
```

A `::=` szimbólum jelentése: „úgy fejezhető ki, hogy”. Az első szabály azt mondja ki, hogy a lekérdezés lehet egy `SELECT…FROM…WHERE` alakú kifejezés. Az `<SFW>`-t leíró szabályokat a későbbiekben adjuk meg. A teljes SQL-nyelvtanban olyan szabályokra is szükség van, amelyek lehetővé teszik, hogy a lekérdezés olyan összetett kifejezés legyen, amely különböző műveleteket (például `UNION`) tartalmaz.

Az `<SFW>` kategóriát egyetlen szabállyal definiáljuk:

```
<SFW> ::= SELECT <SelLista> FROM <FromLista> WHERE <Feltétel>
```

Ez a szabály a `SELECT` utasítás egy korlátozott formáját engedi meg. Nem gondoskodik sem a különböző opcionális záradékokról (`GROUP BY`, `HAVING`, `ORDER BY`), sem egyéb opciókról, mint például a `DISTINCT`, és kötelezőnek tekinti a `WHERE` ágat. Az általunk alkalmazott konvenció szerint a kulcsszavak nagybetűsek.

```
<SelLista> ::= <Attribútum>, <SelLista>
<SelLista> ::= <Attribútum>
```

Ezek a szabályok azt fogalmazzák meg, hogy egy select-lista attribútumoknak vesszővel elválasztott tetszőleges listája lehet, vagy egyetlen attribútum. A teljes SQL-nyelvtanban biztosítani kell a kifejezések használatának és az átnevezésnek a lehetőségét.

```
<FromLista> ::= <Reláció>, <FromLista>
<FromLista> ::= <Reláció>
```

Ezek alapján a from-lista relációknak vesszővel elválasztott tetszőleges listája lehet, vagy egyetlen reláció. Elhagytuk azt a lehetőséget, hogy egy from-lista elemei lehetnek `R JOIN S` vagy `SELECT…FROM…WHERE` alakú kifejezések, nem biztosítottuk a listában szereplő relációk átnevezési lehetőségét, és nem engedtük meg azt sem, hogy sorváltozót rendeljünk az egyes relációkhoz.

```
<Feltétel> ::= <Feltétel> AND <Feltétel>
<Feltétel> ::= <Sor> IN ( <Lekérdezés> )
<Feltétel> ::= <Attribútum> = <Attribútum>
<Feltétel> ::= <Attribútum> LIKE <Minta>
```

Habár a feltételekhez több szabályt soroltunk fel, mint a többi kategóriához, ezek a szabályok a feltételek lehetséges alakjainak csak töredékét adják meg. Elhagytuk az `OR`, `NOT` és `EXISTS` operátorok bevezetését jelentő szabályokat, a legtöbb relációs operátort, a kifejezés operandusokat és számos egyéb struktúrát (például az `IN` operátor második operandusa nem csak lekérdezés lehet).

```
<Sor> ::= <Attribútum>
```

Az egyszerűség kedvéért a sor nálunk csak egyetlen attribútumból állhat, a teljes SQL-nyelvtanban természetesen többféle formában megjelenhet.

Az `<Attribútum>`, a `<Reláció>` és a `<Minta>` speciális szintaktikus kategóriák, mivel ezeket nem nyelvtani szabályok definiálják, hanem az olyan atomokra vonatkozó szabályok, amelyek helyén szerepelnek. Egy elemzőfában az `<Attribútum>` egyetlen gyereke például egy olyan karakterlánc lehet, amely egy attribútum neveként értelmezhető abban az adatbázissémában, amelyre a lekérdezés vonatkozik. Hasonlóképpen a `<Reláció>` egy olyan karakterlánccal helyettesíthető, amely relációnevet jelent az adott sémában, a `<Minta>` pedig egy tetszőleges karakteres kifejezés lehet.

Példa. Az elemzési és lekérdezésátírási fázisok tanulmányozásához a következő relációkat és egy rájuk vonatkozó lekérdezés két változatát fogjuk használni:

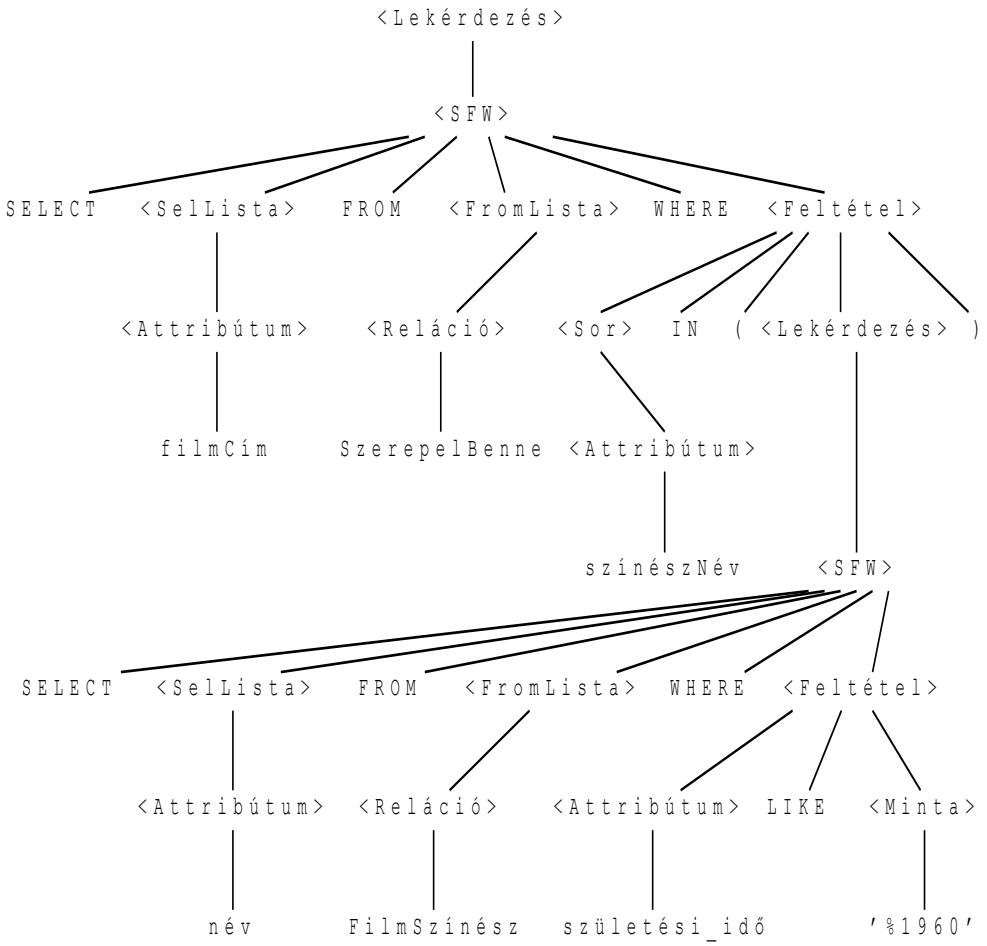
```
SzerepelBenne(filmCím, év, színészNév)
FilmSzínész(név, cím, nem, születési_idő)

SELECT filmCím FROM SzerepelBenne WHERE színészNév IN
  (SELECT név FROM FilmSzínész WHERE születési_idő LIKE '%1960');

SELECT filmCím FROM SzerepelBenne, FilmSzínész
  WHERE színészNév = név AND születési_idő LIKE '%1960';
```

A lekérdezés mindkét változata azoknak a filmeknek a címét kéri, amelyekben legalább egy 1960-ban született színész szerepel. A színészek 1960-as születését úgy állapítjuk meg, hogy a `LIKE` operátor segítségével megvizsgáljuk, hogy a születési idő (karakterlánccá konvertálva) `'1960'`-ra végződik-e.

A lekérdezés megfogalmazásának egyik módja, hogy egy alkérdéssel felépítjük azon színészek neveinek halmazát, akik 1960-ban születtek, majd minden egyes `SzerepelBenne` sorra megvizsgáljuk, hogy benne a `színészNév` eleme-e az alkérdés által visszaadott halmaznak. A másik formában nem használunk alkérdést, hanem helyette összekapcsoljuk a `SzerepelBenne` és a `FilmSzínész` relációkat egy equijoinnal, így megkapjuk mindkét relációból az összes attribútumot, és azokat a sorokat, amelyekben a színész ugyanaz.

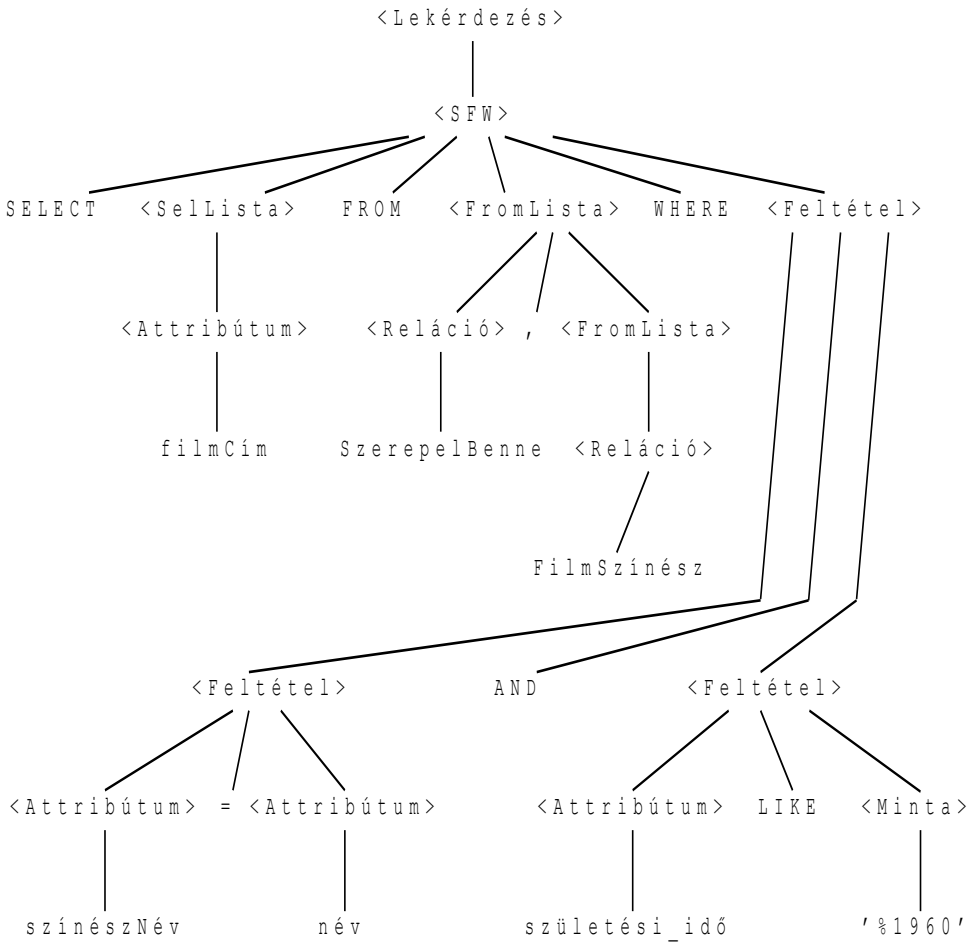
Mindkét lekérdezés ismétlődéseket állít elő akkor, ha valamely filmnek több olyan szereplője is van, aki 1960-ban született. Ahhoz, hogy biztosítsuk az egyediséget, egy `DISTINCT` kulcsszót is meg kellene adni, de a példa nyelvtanunkat oly mértékben leegyszerűsítettük, hogy nem szerepel benne ez az opció.

Az első lekérdezéshez tartozó elemzőfát a következő ábra mutatja:



A gyökérben a `<Lekérdezés>` szintaktikus kategória áll, aminek minden elemzőfa esetében így kell lennie. Lefelé haladva a fában azt látjuk, hogy ez egy `SELECT…FROM…WHERE` alakú lekérdezés, amelynek select-listája csak a `filmCím` attribútumból áll, és a from-lista csak a `SzerepelBenne` relációt tartalmazza. A külső `WHERE` záradék feltétele már összetettebb. `Sor IN (Lekérdezés)` alakú, ahol a lekérdezés zárójelezett, mivel az SQL-ben az alkérdéseket zárójelek közé kell tenni. Maga az alkérdés egy újabb `SELECT…FROM…WHERE` kifejezés a saját egyelemű select- és from-listájával és egy egyszerű feltétellel, amely a `LIKE` operátort használja.

A következő ábra szemlélteti a második lekérdezéshez tartozó elemzőfát:



Az ebben az elemzőfában használt szabályok közül sok ugyanaz, mint az első elemzőfánál használtak. Figyeljük meg azonban, hogy a több relációt tartalmazó from-listát hogyan fejezi ki az elemzőfa, valamint azt is megfigyelhetjük ebben az esetben, hogy egy feltétel hogyan áll össze több kisebb feltételből az `AND` operátor segítségével.

Az előfeldolgozó

Az *előfeldolgozónak* (preprocessor) több fontos funkciója van. Ha a lekérdezésben használt valamely reláció egy nézettábla, akkor a relációt a nézettáblának megfelelő elemzőfával kell helyettesíteni, valahányszor a from-listában szerepel. Ezt az elemzőfát a nézettábla definíciója alapján kapjuk meg, ami valójában egy lekérdezés.

Az előfeldolgozó *szemantikai ellenőrzések* (semantic checking) elvégzéséért is felelős. Még ha érvényes is a lekérdezés szintaktikai szempontból, esetleg megsérthet bizonyos szabályokat a nevek használatára vonatkozóan. Az előfeldolgozónak például az alábbiakat kell elvégezni:

1. *Relációk használatának ellenőrzése*: A `FROM` záradékban szerepeltetett relációk mindegyike annak a sémának egy relációja vagy nézettáblája kell, hogy legyen, amelyre a lekérdezés vonatkozik. A fenti első példában az előfeldolgozó ellenőrizni fogja, hogy a két from-listában megadott `SzerepelBenne` és `FilmSzínész` relációk valóban a séma relációi-e.
2. *Attribútumnevek ellenőrzése és feloldása*: A `SELECT` vagy `WHERE` záradékokban előforduló attribútumok mindegyike az aktuális érvényességi kör valamely relációjának egy attribútuma kell, hogy legyen. A fenti első példa első select-listájában például a `filmCím` attribútum csak a

`SzerepelBenne` reláció hatáskörébe esik. Szerencsére a `SzerepelBenne` relációnak attribútuma a `filmCím`, így az előfeldolgozó itt jóváhagyja a `filmCím` használatát. Ezen a ponton a tipikus lekérdezésfeldolgozó minden egyes attribútum feloldását elvégezné – összekapcsolva a megfelelő relációval –, feltéve hogy ez nem történt meg explicit módon a lekérdezésben (például `SzerepelBenne.filmCím`). Ellenőrizné továbbá az egyértelműséget is, és hibát jelezne, ha a vizsgált attribútum több relációban szerepelne attribútumként az érvényességi körön belül.

3. *Típusellenőrzés*: Minden attribútum csak a használatának megfelelő típusú lehet. A fenti első példában például a `születési_idő`-t egy `LIKE` összehasonlításban használjuk, ami megköveteli, hogy a `születési_idő` karakterlánc legyen, vagy olyan típusú, amely karakterlánccá konvertálható. Mivel a `születési_idő` egy dátum, és az SQL-ben a dátumokat kezelhetjük karakterláncokként, az attribútumnak ez a használata elfogadható. Ehhez hasonlóan azt is ellenőrizni kell, hogy operátorokat csak megfelelő típusú értékekre alkalmazunk-e.

Sikeresen túljutva ezeken az ellenőrzéseken, az elemzőfáról azt mondjuk, hogy *érvényes*, és miután megtörtént az esetleges nézettáblák kifejtése és az attribútumnevek használatának feloldása, a fát továbbadjuk a logikai lekérdezéstervet generálónak. Ha az elemzőfa nem érvényes, akkor megfelelő diagnosztika készül, és a feldolgozás leáll.

Algebrai megközelítés

Ahhoz, hogy a lekérdezések végrehajtására szolgáló jó algoritmusokról beszélhessünk, először meg kell adnunk a lekérdezéseket alkotó elemi relációalgebrai műveleteket. Számos SQL-lekérdezés kifejezhető a „klasszikus” relációalgebrát alkotó néhány operátorral, ám a legtöbb lekérdezőnyelvnek (így az SQL-nek is) vannak olyan lehetőségei, amelyekhez azok nem elegendőek. Éppen ezért most egy „kibővített” realációalgebrát ismertetünk, olyan operátorokkal, mint például a csoportosítás, a rendezés és az ismétlődések kiküszöbölése.

A relációalgebrát eredetileg úgy tervezték, mintha a relációk halmazok lennének. Az SQL-ben azonban a relációk *multihalmazok*. Ez azt jelenti, hogy ugyanaz a sor többször is megjelenhet egy relációban. (Bár a matematikai multihalmazokban az elemek sorrendisége nem értelmezett, a relációkban viszont bizonyos esetekben igen, jobb szó híján multihalmazoknak tekintjük a relációkat.) A másik fontos különbség a „klasszikus” relációalgebrához képest, hogy a relációs séma attribútumok halmaza, azaz az attribútumok sorrendje nem számít.

A relációalgebrai operátorok a következők (zárójelben az SQL-beli megfelelőikkel):

- egyesítés vagy unió, metszet, különbség (`UNION`, `INTERSECT`, `EXCEPT`);
- kiválasztás vagy szelekció (`WHERE`, `HAVING`);
- vetítés vagy projekció (`SELECT`);
- szorzat (`FROM`);
- összekapcsolás (`JOIN`, `NATURAL JOIN`, `OUTER JOIN` vagy `FROM+WHERE`);
- ismétlődések kiküszöbölése (`DISTINCT`);
- csoportosítás (`GROUP BY`);
- rendezés (`ORDER BY`).

Egyesítés, metszet és különbség

Ha a relációk halmazok lennének, akkor a $\cup$, $\cap$ és $-$ operátorok hatása megegyezne a megfelelő matematikai műveletek hatásával. Van azonban két fontos különbség a relációk és a halmazok között:

a) A relációk multihalmazok.
b) A relációk rendelkeznek sémával, ami nem más, mint az oszlopok neveinek megfelelő attribútumhalmaz.

A b) problémával könnyű megbirkózni. Mindhárom műveletnél megköveteljük, hogy a két argumentum reláció sémája megegyezzen (*unió-kompatibilitás*). Ily módon az eredményül kapott reláció sémája ugyanaz lesz, mint az argumentumoké.

Az a) probléma sem okoz gondot, csak újra kell definiálni a három műveletet multihalmazokra:

1. $R \cup S$ esetén egy t sor annyiszor fordul elő az eredményben, amennyi t R-ben és S-ben lévő előfordulásainak az összege, azaz ahányszor előfordul R-ben és S-ben együttvéve.
2. $R \cap S$ esetén egy t sor annyiszor fordul elő az eredményben, amennyi t R-ben és S-ben lévő előfordulásainak a minimuma.
3. $R - S$ esetén egy t sor annyiszor fordul elő az eredményben, amennyi t R-ben és S-ben lévő előfordulásainak a különbsége, illetve ha ez a különbség negatív, akkor nullaszor.

Figyeljük meg, hogy ha R és S történetesen halmazok, vagyis egyikben sem jelenik meg egy elem kétszer, akkor az $R \cap S$ és az $R - S$ műveletek eredménye pontosan ugyanaz, mint amit a halmazokon értelmezett operátorok adnának. A multihalmazos $R \cup S$ műveletnek azonban ekkor is lehet olyan végeredménye, ami nem halmaz.

Példa. Legyen $R = \{A,B,B\}$, $S = \{C,A,B,C\}$ két multihalmaz. Ekkor:

- $R \cup S = \{A,A,B,B,B,C,C\}$
- $R \cap S = \{A,B\}$
- $R - S = \{B\}$

Az egyesítés elemeit rendezetten tüntettük fel, de ne feledjük, hogy a sorrend nem számít. A multihalmazokon végzett műveletek fenti szabályai függetlenek attól, hogy R és S elemei sorok, objektumok vagy valami mások. A relációalgebránál feltételezzük, hogy R, S és az eredmény relációk, elemeik pedig sorok.

Alapértelmezésben a `UNION`, az `INTERSECT` és az `EXCEPT` SQL-műveletek kiküszöbölik az eredményből az ismétlődéseket, még akkor is, ha az argumentum relációk tartalmaznak ismétlődéseket. Ezeknek a műveleteknek a multihalmazos változatát SQL-ben az `ALL` kulcsszó segítségével képezhetjük, például `UNION ALL`. Elképzelhető olyan SQL-megvalósítás is, ahol a fenti műveletek eleve multihalmaz alapúak, amelyeket követ az ismétlődések kiküszöbölésére szolgáló művelet.

A félreértések elkerülése érdekében úgy különböztetjük meg a kétfajta operátortípust, hogy alsó indexben jelöljük az egyes operátorok típusát: S-sel jelöljük a halmaz (set), B-vel a multihalmaz (bag) alapú műveleteket. Így például $\cup_S$ a halmaz alapú egyesítés, $-_B$ a multihalmazos különbség. Ha nem írunk alsó indexet, akkor alapértelmezés a multihalmazos változat.

Beszélnünk kell még egy speciális műveletről, ez pedig a *külső egyesítés* (outer union). A külső egyesítésnek nem előfeltétele az unió-kompatibilitás, csak a *részleges kompatibilitás*, ami azt jelenti, hogy a két relációnak csak néhány attribútuma unió-kompatibilis. (Halmaz alapú külső egyesítés esetén további feltétel, hogy a kompatibilis attribútumok listáinak tartalmaznia kell egy kulcsot mindkét relációban.) Két reláció (R és S) külső egyesítésén egy olyan relációt értünk, amelynek sémája a két reláció sémájának uniója, elemeit pedig úgy kapjuk meg, hogy vesszük R elemeit, S attribútumain `NULL` értékekkel kiegészítve, valamint S elemeit, R attribútumain `NULL` értékekkel kiegészítve. Ezt a műveletet az `OUTER UNION` művelettel valósíthatjuk meg, bár ez a művelet nem szerepel az SQL-szabványban. Akárcsak a `UNION`, az `OUTER UNION` is kiküszöböli az eredményből az ismétlődéseket.

Az általános külső egyesítés annyiban tér el a fent ismertetett (Codd-féle) külső egyesítéstől, hogy még a részleges kompatibilitást sem követeljük meg, és nem küszöböli ki az eredményből az ismétlődéseket.

Példa. Legyen `R(a,b)` a következő reláció:

a	b
0	1
2	3
2	3

Az `S(b,c)` reláció pedig legyen a következő:

b	c
1	4
1	4
2	5

Ekkor `R` és `S` általános külső egyesítésének eredménye a következő reláció lesz:

a	b	c
0	1	NULL
2	3	NULL
2	3	NULL
NULL	1	4
NULL	1	4
NULL	2	5

Kiválasztás

A $\sigma_C(R)$ *kiválasztás* (selection) tartalmaz egy `R` relációt és egy `C` feltételt. A `C` feltétel magában foglalhat:

1. konstansokra és/vagy attribútumokra alkalmazott aritmetikai (például `+`, `*`) vagy karakteres (például `||`, `LIKE`) operátorokat;
2. az 1. segítségével felépített kifejezések összehasonlítását (például `a < b` vagy `a + b = 10`);
3. a 2. segítségével felépített kifejezésekre alkalmazott `AND`, `OR` és `NOT` logikai összekapcsolásokat (például `a < b AND a + b = 10`).

Az eredmény reláció sémája megegyezik `R`-ével. A $\sigma_C(R)$ kiválasztás az `R` reláció azon sorainak multihalmazát eredményezi, amelyek kielégítik a `C` feltételt.

Példa. Legyen `R(a,b)` a következő reláció:

a	b
0	1
2	3
4	5
2	3

A $\sigma_{a \geq 1}(R)$ kiválasztás eredménye:

a	b
2	3
4	5
2	3

Figyeljük meg, hogy egy olyan sor, amelyik kielégíti a feltételt, ugyanannyiszor jelenik meg az eredményben, mint magában a relációban, míg egy olyan sor, amelyik nem teljesíti a feltételt, egyáltalán nem jelenik meg.

A $\sigma_{b \geq 3 \text{ AND } a + b \geq 6}(R)$ kiválasztás eredménye:

a	b
4	5

Az itt bevezetett σ operátor sokkal erőteljesebb, mint a klasszikus relációalgebra kiválasztás operátora, mivel megengedtük a feltételben az OR és a NOT logikai operátorokat is. Azonban még ez a σ operátor sincs olyan erőteljes, mint a WHERE záradék az SQL-ben, mivel ott szerepelhetnek alkérdések és relációkra értelmezett logikai operátorok, mint például az EXISTS. Egy alkérdést tartalmazó feltételt teljes relációkon dolgozó operátorral kell kifejeznünk, míg a mi kiválasztásunk feltétele csak konkrét sorok tesztelésére alkalmazható. A későbbiekben olyan kiválasztás operátorokról is fogunk beszélni, amelyek engedélyezik az alkérdéseket mint explicit argumentumokat.

Vetítés

Ha R egy reláció, akkor $\pi_L(R)$ az R reláció L listára történő *vetítése* (projection). A klasszikus relációalgebrában L az R reláció attribútumainak egy listája. Kiterjesztjük a vetítés operátort, hogy hasonlóvá váljon az SQL SELECT záradékához. A mi vetítési listánk a következő típusú elemeket tartalmazhatja:

1. R egy attribútumát.
2. Egy $x \rightarrow y$ kifejezést, ahol x és y attribútumnevek. Az $x \rightarrow y$ kifejezés az L listában azt jelenti, hogy vesszük az R reláció x attribútumát, és átnevezzük y-ra, azaz az eredmény relációban ennek az attribútumnak a neve y lesz.
3. Egy $E \rightarrow z$ kifejezést, ahol E az R reláció attribútumait, konstansokat, aritmetikai operátorokat és karakteres operátorokat tartalmazó kifejezés, z pedig az új neve annak az attribútumnak, amelyet az E-ben szereplő számítások eredményeznek. Például az $a + b \rightarrow x$ – mint a lista egy eleme – az a és b attribútumok összegét reprezentálja x néven. A $c \mid\mid d \rightarrow e$ elem a c és d elemek összefűzését jelenti e néven.

Ahhoz, hogy a transzformációkat a vetítés általános alakjával írhassuk le, be kell vezetnünk néhány fogalmat. Legyen $E \rightarrow x$ egy kifejezés egy vetítési listában, ahol E vagy egy attribútum, vagy egy attribútumokat és konstansokat tartalmazó kifejezés! Ekkor az E-ben előforduló összes attribútum a vetítés *bemeneti* attribútuma, x pedig egy *kimeneti* attribútum. Ha pedig a listaelemként szereplő kifejezés egyetlen attribútum, akkor az egyben bemeneti és kimeneti attribútum is. Ha a vetítési lista csak attribútumokból áll, tehát nincs sem átnevezés, sem pedig olyan kifejezés, ami nem egyetlen attribútum, akkor *egyszerű* vetítésről beszélünk. A klasszikus relációalgebrában minden vetítés egyszerű. A $\pi_L(R)$ vetítés *triviális*, ha egyszerű, és L az R reláció összes attribútumát tartalmazza.

Az eredmény reláció sémája a vetítési listában szereplő kimeneti attribútumok halmaza. A vetítés eredményét úgy határozzuk meg, hogy vesszük sorban R valamennyi sorát. Kiértékeljük az L listát oly módon, hogy behelyettesítjük a sorok komponenseit az L-ben felsorolt megfelelő bemeneti attribútumokba, alkalmazzuk a megadott operátorokat ezekre az értékekre, majd az eredményeket beírjuk a megfelelő kimeneti attribútumokba. R valamennyi sora létrehoz egy sort az eredményben. R ismétlődő sorai természetesen ismétlődő sorokat hoznak létre az eredményben, de az eredmény akkor is tartalmazhat ismétlődő sorokat, ha R nem tartalmazott ilyet.

Példa. Legyen R(a,b,c) a következő reláció:

a	b	c
0	1	2
0	1	2
3	4	5

A $\pi_{a,\ b+c \to x}(R)$ eredménye:

a	x
0	3
0	3
3	9

Az eredmény sémája két attribútumot tartalmaz. Az egyik a, amely R első attribútuma, a másik R második és harmadik attribútumának összege x néven.

A $\pi_{b-a \to x,\ c-b \to y}(R)$ eredménye:

x	y
1	1
1	1
1	1

Figyeljük meg, hogy a vetítési listában megadott számítások történetesen ugyanazt az (1, 1) sort eredményezték a (0, 1, 2) sorra és a (3, 4, 5) sorra egyaránt. Ezért az (1, 1) sor háromszor jelenik meg az eredményben.

Példa. A $\pi_{a,\ b,\ c}(R)$ vetítés egyszerű; a, b és c egyszerre bemeneti és kimeneti attribútumok. A $\pi_{a+b \to x,\ c}(R)$ vetítés viszont nem egyszerű; ennek bemeneti attribútumai az a, a b és a c, kimeneti attribútumai pedig az x és a c.

Szorzat

Ha R és S két reláció, akkor az $R \times S$ *szorzat* (Cartesian product) egy olyan reláció, amelynek sémája R és S attribútumaiból áll. Ha mindkét sémában van a nevű attribútum, akkor a szorzat sémájában az attribútumok neveiként az R.a, illetve az S.a jelöléseket használjuk.

A szorzat sorait az összes olyan sor alkotja, amelyet úgy kapunk, hogy vesszük R egy sorát, és annak elemeit kiegészítjük S egy sorával. Ha egy r sor n-szer jelenik meg R-ben, s pedig m-szer szerepel S-ben, akkor a szorzatban az rs sor nm-szer jelenik meg.

Példa. Legyen R(a,b) a következő reláció:

a	b
0	1
2	3
2	3

Az S(b,c) reláció pedig legyen a következő:

b	c
1	4
1	4
2	5

Ekkor R × S eredménye a következő reláció lesz:

a	R.b	S.b	c
0	1	1	4
0	1	1	4
0	1	2	5
2	3	1	4
2	3	1	4
2	3	2	5
2	3	1	4
2	3	1	4
2	3	2	5

Figyeljük meg, hogy R valamennyi sorát párosítottuk S valamennyi sorával, tekintet nélkül az ismétlődésekre. Így például a (2, 3, 1, 4) sor négyszer jelenik meg, mivel az őt alkotó komponensek az R, illetve az S reláció ismétlődő sorai.

Összekapcsolások

Számos olyan hasznos *összekapcsolás* (join) operátor van, amely egy szorzatból és az azt követő kiválasztásból és vetítésből épül fel. Ezek az operátorok explicit módon megtalálhatók az SQL2-szabványban, mint a FROM záradékban felsorolt relációk kombinálásának módjai. Összekapcsolás azonban az olyan lekérdezés is, amelynek FROM záradéka egy vagy több relációt tartalmaz, és amelynek WHERE záradékában ezen relációk attribútumaira alkalmazott összehasonlítások szerepelnek.

A legegyszerűbb és legelterjedtebb a *természetes összekapcsolás* (natural join). Az R és S relációk természetes összekapcsolását az R * S szimbólummal jelöljük. Ez a kifejezés a $\pi_L(\sigma_C(R \times S))$ rövidítése, ahol:

- C egy olyan feltétel, amely megköveteli R és S azonos nevű attribútumainak (azaz az *összekapcsolási attribútumok*) páronkénti egyenlőségét AND operátorral összekapcsolva;
- L egy attribútumlista, amely R és S összes attribútumát tartalmazza, kivéve az azonos nevű attribútumokat, amelyek csak egy példányban szerepelnek ebben a listában. Ha R.x és S.x a két egyenlővé tett attribútum, akkor a vetítés eredményében csak egy x attribútum fog szerepelni.

Példa. Ha R(a,b) és S(b,c) az előző példában bevezetett relációk, akkor R * S a $\pi_{a,\ R.b \to b,\ c}(\sigma_{R.b = S.b}(R \times S))$ kifejezést jelenti. Ez azt jelenti, hogy mivel az R és S relációkban a b az egyetlen közös attribútum, ezért a kiválasztás csak ezt a két attribútumot teszi egyenlővé. Most az R.b-t választottuk, és ezt neveztük át b-re, de ugyanígy választhattuk volna S.b-t is.

Egy természetes összekapcsolás eredményét megkaphatjuk úgy is, hogy sorban alkalmazzuk a ×, σ és π operátorokat. Könnyebb azonban „egy lépésben” kiszámolni a természetes összekapcsolást. Számos módszer létezik arra vonatkozóan, hogy miként találjuk meg az R és S relációk azon sorpárjait, amelyek megegyeznek az összes azonos nevű attribútumon. Ezekre a sorpárokra képezzük az eredmény sorokat, amelyek minden attribútumon megegyeznek ezekkel a sorokkal. A fenti példában például azon sorpárok, amelyekre b értéke megegyezik, az R (0, 1) és az S két (1, 4) sorából tevődnek össze. R * S tehát:

a	b	c
0	1	4
0	1	4

Figyeljük meg, hogy két összekapcsolandó sorpár van, amelyek történetesen S azonos sorait tartalmazzák. Ez az oka annak, hogy az eredményben kétszer jelenik meg a (0, 1, 4) sor.

Az összekapcsolás másik formája a *théta-összekapcsolás* (theta join). Az R és S relációk esetén az $R \otimes_C S$ a $\sigma_C(R \times S)$ kifejezés rövidítése, ahol C tartalmaz legalább egy $x\ \theta\ y$ alakú tagot, ahol x az R, y pedig az S reláció attribútumaiból (az *összekapcsolási attribútumokból*) álló kifejezés, a θ pedig egy tetszőleges összehasonlító operátor. Ha a C feltétel csak olyan tagokat tartalmaz (AND operátorral összekapcsolva), amelyek $x = y$ alakúak, ahol x az R, y pedig az S reláció attribútuma, akkor ezt az összekapcsolást *egyenlőség alapú összekapcsolásnak* (equijoin) nevezzük. A természetes összekapcsolással ellentétben az egyenlőség alapú összekapcsolás nem tartalmaz vetítést, még akkor sem, ha az eredményben két vagy több egyforma oszlop szerepel.

Példa. Legyenek R(a,b) és S(b,c) a fenti példában bevezetett relációk. Ekkor az $R \otimes_{a+R.b<c+S.b} S$ megegyezik a $\sigma_{a+R.b<c+S.b}(R \times S)$ kifejezéssel. Ez azt jelenti, hogy az összekapcsolás feltétele megköveteli, hogy R sorában a komponensek összege kisebb legyen, mint S sorában a komponensek összege. Az eredmény tartalmazza $R \times S$ összes sorát, kivéve azt, amelyben R sora (2, 3), S sora pedig (1, 4), mivel itt az R-beli összeg nem kisebb, mint az S-beli összeg. Az eredmény reláció így a következő:

a	R.b	S.b	c
0	1	1	4
0	1	1	4
0	1	2	5
2	3	2	5
2	3	2	5

Legyen egy másik példa az $R \otimes_{b=b} S$ egyenlőség alapú összekapcsolás kiszámítása. Megállapodás szerint a théta-összekapcsolásban egyenlővé tett attribútumok közül az első a bal oldali argumentumhoz tartozik, míg a második a jobb oldali argumentumhoz. A fenti kifejezés tehát ugyanaz, mint $R \otimes_{R.b=S.b} S$. Az eredmény ugyanaz, mint az R * S eredménye, kivéve, hogy mindkét egyenlővé tett attribútum megmarad:

a	R.b	S.b	c
0	1	1	4
0	1	1	4

A természetes, a théta- és az egyenlőség alapú összekapcsoláson kívül a következő speciális összekapcsolásokat definiálhatjuk:

1. Az R és S relációk *félig összekapcsolása* (semijoin) szűk értelemben az R reláció azon t sorainak multihalmaza, amelyekre létezik legalább egy olyan sor S-ben, amely megegyezik t-vel R és S valamennyi közös attribútumán. Tágabb értelemben az R reláció azon sorainak multihalmazát jelenti, amelyeknek van párjuk az S relációban az adott théta-összekapcsolás feltétele alapján.
2. Az R és S relációk *félig antiösszekapcsolása* (antisemijoin) szűk értelemben az R reláció azon t sorainak multihalmaza, amelyekre nem létezik egy olyan sor sem S-ben, amely megegyezik t-vel R és S valamennyi közös attribútumán. Tágabb értelemben az R reláció azon sorainak multihalmazát jelenti, amelyeknek nincs párjuk az S relációban az adott théta-összekapcsolás feltétele alapján. Ezeket a sorokat az R reláció *lógó sorainak* (dangling tuples) nevezzük.
3. Az R és S relációk *(teljes) külső összekapcsolása* (full outer join) az $R \otimes_C S$ soraiból áll, amelyekhez még hozzávesszük R és S lógó sorait. Az ily módon hozzáadott sorokat ki kell egészítenünk NULL értékekkel mindazon attribútumok helyén, amelyekkel ezek a sorok nem rendelkeznek, de az eredmény sorok igen.

4. Az R és S relációk *bal oldali külső összekapcsolása* (left outer join) olyan, mint a külső összekapcsolás, de most csak R lógó sorait egészítjük ki NULL értékekkel, és adjuk hozzá az eredményhez.
5. Az R és S relációk *jobb oldali külső összekapcsolása* (right outer join) olyan, mint a külső összekapcsolás, de most csak S lógó sorait egészítjük ki NULL értékekkel, és adjuk hozzá az eredményhez.

Ismétlődések kiküszöbölése

Szükségünk van egy olyan operátorra, amely egy multihalmazt halmazzá alakít az SQL DISTINCT kulcsszavának megfelelően. Erre a célra a $\delta(R)$ operátort használjuk, amely visszaadja azt a halmazt, amely az R relációban előforduló sorokból egyetlen példányt tartalmaz. Az eredmény reláció sémája megegyezik R sémájával.

Példa. Ha az R reláció megfelel az előző példák ugyanilyen nevű relációjának, akkor $\delta(R)$ eredménye:

a	b
0	1
2	3

Figyeljük meg, hogy a (2, 3) sor, amely kétszer jelent meg az R relációban, $\delta(R)$-ben csak egyszer szerepel.

Emlékezzünk vissza, hogy az SQL UNION, INTERSECT és EXCEPT operátorai alapértelmezés szerint kiküszöbölik az ismétlődéseket, viszont mi úgy definiáltuk az $\cup$, $\cap$ és $-$ operátorokat, hogy megfeleljenek az alapértelmezés szerinti multihalmazos meghatározásnak. Ezért ha egy olyan SQL-kifejezést szeretnénk algebrai kifejezéssé alakítani, mint az R UNION S, akkor azt kell írnunk, hogy $\delta(R \cup S)$ (vagy $R \cup_S S$).

Csoportosítás és összesítés

Az SQL-ben egész sor lehetőség van az olyan lekérdezések támogatására, amelyek „*csoportosítást és összesítést*” (grouping and aggregation) használnak:

1. *Összesítő operátorok* (aggregation operators). Az öt operátor, az AVG, a SUM, a COUNT, a MIN és a MAX annak a kifejezésnek az átlagát, összegét, a benne található elemek számát, minimumát, illetve maximumát határozzák meg, amelyekre alkalmazzuk őket. Ezek az operátorok a SELECT záradékokban jelennek meg.
2. *Csoportosítás* (grouping). Egy SQL-lekérdezés GROUP BY záradéka által a FROM és WHERE záradékok alapján felépített reláció csoportosítva lesz a GROUP BY záradékban felsorolt attribútumok alapján. Ezek után az összesítések a csoportokra készülnek el.
3. Egy HAVING záradékot kötelezően egy GROUP BY záradéknak kell megelőznie, és egy olyan feltételt fogalmaz meg (ez a feltétel érinthet összesítéseket és a csoportosítás alapját képező attribútumokat is), amelyet egy csoportnak teljesítenie kell ahhoz, hogy a lekérdezés eredményének részét képezze.

A csoportosításokat és az összesítéseket általában együtt kell megvalósítani és optimalizálni. Ily módon egyetlen olyan γ operátort fogunk bevezetni a kiterjesztett relációalgebránkba, amely a csoportosítás és az összesítés hatását reprezentálja. A γ operátor segít a HAVING záradék megvalósításában is, amelyet a γ-t követő kiválasztás és vetítés képvisel.

A γ operátor alsó indexét egy `L` lista képezi, amelynek valamennyi eleme a következők egyike:

a) A reláció egy attribútuma, amelyre a γ-t alkalmazzuk; ez az attribútum egyike a lekérdezés `GROUP BY` listájának. Ezt az elemet *csoportosító* (grouping) attribútumnak nevezzük.
b) A reláció attribútumaiból képzett kifejezésre alkalmazott összesítő operátor. Ahhoz, hogy az eredményben névvel lássuk el az összesítés eredményeként előálló attribútumot, egy nyilat és egy új nevet írunk az összesítés után. Ez az elem a lekérdezés `SELECT` záradékának egyik összesítését reprezentálja. Az eredmény reláció megfelelő attribútumát *összesített* (aggregated) attribútumnak nevezzük.

A $\gamma_{L}(R)$ kifejezés által visszaadott reláció a következőképpen épül fel:

1. `R` sorait csoportokba osztjuk szét. Valamennyi csoport azokból a sorokból épül fel, amelyek az `L` lista csoportosító attribútumaira egy bizonyos értékkel rendelkeznek. Ha nincsenek csoportosító attribútumok, akkor a teljes `R` reláció lesz egy csoport.
2. Minden egyes csoportra képezünk egy olyan sort, amely a következőket tartalmazza:
   i) a csoportosító attribútumok értékeit az adott csoportra;
   ii) a csoport összes sorára vonatkozó összesítéseket, amelyeket az `L` lista összesített attribútumai specifikálnak.

Példa. Tegyük fel, hogy szeretnénk megkapni azon színészeket, akik legalább három filmben szerepeltek, azzal az évszámmal együtt, amikor először szerepeltek. A választ a következő SQL-lekérdezés adja meg:

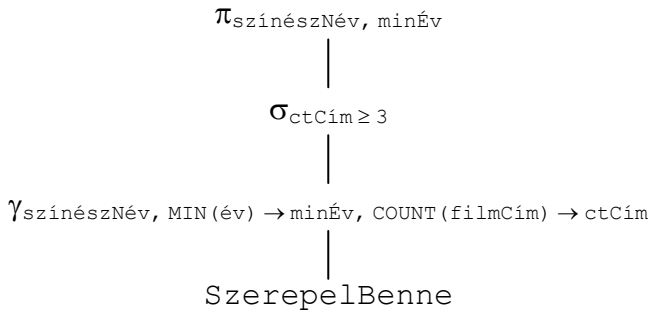
```
SELECT színészNév, MIN(év) AS minÉv FROM SzerepelBenne
  GROUP BY színészNév HAVING COUNT(filmCím) >= 3;
```

Az ekvivalens algebrai kifejezés csoportosítást fog végezni a `színészNév` attribútumon. Minden bizonnyal ki kell számolnunk minden csoportra a `MIN(év)` összesítést. Ahhoz azonban, hogy meg tudjuk ítélni, hogy melyik csoport teljesíti a `HAVING` záradékot, ki kell számolnunk a `COUNT(filmCím)` összesítést is valamennyi csoportra.

A csoportosító kifejezés a következő lesz:

$$\gamma_{\texttt{színészNév, MIN(év)} \rightarrow \texttt{minÉv, COUNT(filmCím)} \rightarrow \texttt{ctCím}}(\texttt{SzerepelBenne})$$

A kifejezés eredményének első két oszlopára a lekérdezés eredménye miatt van szükség. A harmadik oszlop egy kiegészítő attribútum, amelyet `ctCím`-nek neveztünk el. Ez azért szükséges, mert minden sorra alkalmaznunk kell a `HAVING` záradékban szereplő feltételt. Ez azt jelenti, hogy a lekérdezéshez tartozó algebrai kifejezést azzal folytatjuk, hogy kiválasztjuk a `ctCím >= 3` feltételnek megfelelő sorokat, majd az eredményt levetítjük az első két oszlopra. A lekérdezés reprezentációja a következő ábrán látható:



Ez egy egyszerűbb formájú kifejezésfa, ahol egymás után négy operátort láthatunk, mindegyik az előző operátor alatt foglal helyet.

Még a HAVING záradék nélküli csoportosító lekérdezések között is van olyan, amelyet nem lehet kifejezni csupán egyetlen γ operátorral. A FROM záradék például több relációt is tartalmazhat, és ezeket először egyesíteni kell egy szorzat operátorral. Ha a lekérdezésnek van egy WHERE záradéka, akkor ennek a záradéknak a feltételét egy σ operátorral ki kell fejezni, vagy esetleg a relációk szorzatát egy összekapcsolássá kell alakítani. Előfordulhat továbbá, hogy a GROUP BY záradék egyik attribútuma nem szerepel a SELECT záradékban. A fenti példában például elhagyhatjuk a színészNév attribútumot a SELECT záradékból, habár a hatás egy kissé furcsa lenne: kapnánk egy évszámokból álló listát, de semmi sem jelölné, hogy melyik év melyik színésznek felel meg. Ebben az esetben a GROUP BY összes attribútumát felsoroljuk a γ listájában, majd ezután alkalmazunk egy vetítést, amely eltávolítja azokat a csoportosító attribútumokat, amelyek nem jelennek meg a SELECT záradékban.

A γ operátor bevezetése után belátható, hogy a δ operátor technikailag redundáns. Ha $R(a_1, a_2, \ldots, a_n)$ egy reláció, akkor $\delta(R)$ ekvivalens a $\gamma_{a1, a2, \ldots, an}(R)$ kifejezéssel. Ez azt jelenti, hogy az ismétlődések kiküszöböléséhez a reláció összes attribútumán csoportosítunk, és nem végzünk összesítést. Így mindegyik csoport megfelel egy olyan sornak, amely egyszer vagy többször szerepel R-ben. Mivel a γ eredménye minden csoporthoz pontosan egy sort tartalmaz, ezért ennek a csoportosításnak az eredménye az, hogy kiküszöböli az ismétlődéseket. Azonban mivel a δ egy igen elterjedt és fontos operátor, külön fogunk vele foglalkozni az algebrai törvényszerűségek, valamint az operátorok megvalósítására szolgáló algoritmusok tárgyalásakor.

Rendezés

A τ operátort fogjuk használni egy reláció rendezéséhez. Ezt az operátort az SQL ORDER BY záradékának megvalósítására lehet használni. A rendezés azonban egy fizikai lekérdezésterv operátorának szerepét is betöltheti, hiszen a relációalgebra sok más operátora gyorsabbá tehető, ha először rendezünk egy vagy több argumentumban szereplő relációt.

A $\tau_L(R)$ kifejezés pontosan az R relációt adja, de sorai rendezettek lesznek az L által megadott módon, ahol R egy reláció, L pedig az R reláció egyes attribútumainak listája. Ha L az $a_1, a_2, \ldots, a_n$ lista, akkor R sorai először az a_1 attribútum értékei szerint lesznek rendezve, egyforma a_1 értékek esetén az a_2 értékei szerint stb. Azok a sorok, amelyek még az a_n attribútumon is megegyeznek, tetszőleges sorrendbe helyezhetők. Éppúgy, mint az SQL, feltételezzük, hogy az alapértelmezett rendezési sorrend növekvő, de ez csökkenőre változtatható az attribútum neve után írt DESC kulcsszóval.

A τ operátor szabálytalan abban a tekintetben, hogy a mi relációalgebránkban ez az egyetlen olyan operátor, amelynek eredménye nem multihalmaz, hanem sorok egy listája. Ily módon egy algebrai kifejezésben csak utolsó operátorként van értelme beszélni a τ operátorról. Ha egy másik relációalgebrai operátort alkalmazunk a τ után, akkor a τ eredményét multihalmazként kezeljük, tehát nem számít a sorok sorrendje. Gyakran használjuk azonban a τ-t fizikai lekérdezéstervekben, amelyek operátorai nem ugyanazok, mint a relációalgebrai operátorok. Sok későbbi operátor veszi hasznát annak, ha egy vagy több argumentum rendezett, és lehet, hogy ők maguk is rendezett eredményt állítanak elő.

Kifejezésfák

Több relációalgebrai operátort használhatunk egyetlen kifejezésben, ha egy vagy több operátor eredményére alkalmazunk egy másik operátort. Ily módon éppúgy, mint bármely más algebra esetén, az operátorok egymás utáni alkalmazását egy *kifejezésfa* (expression tree) formájában rajzolhatjuk fel. A fa leveleit relációk nevei alkotják, a belső csúcsokat pedig olyan operátorok alkotják, amelyek akkor nyernek értelmet, amikor alkalmazzuk a gyermeke vagy gyermekei által reprezentált relációkra.

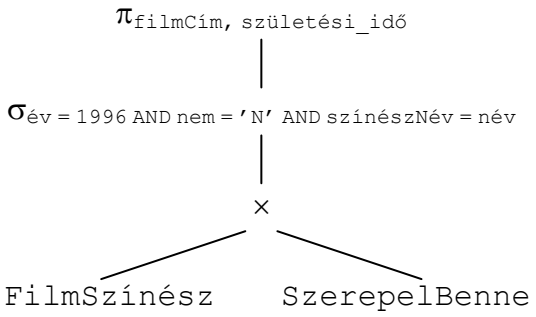
Korábban már láthattunk egy egyszerű kifejezésfát, amelyben három egyoperandusú (unáris) operátort alkalmaztunk egymás után. Sok kifejezésfa tartalmaz azonban kétoperandusú (bináris) operátorokat, amelyeknek két gyermekük van.

Példa. Tegyük fel, hogy szeretnénk kinyerni az 1996-ban megjelent filmek címét a bennük szereplő színésznők születési idejével együtt:

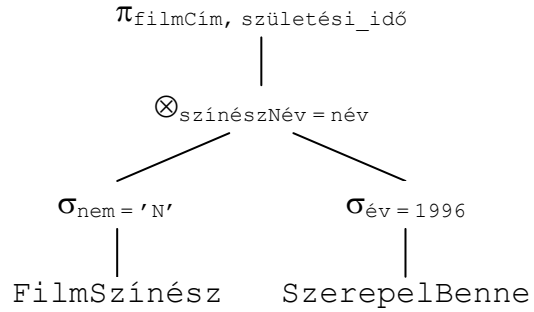
```
SELECT filmCím, születési_idő FROM FilmSzínész, SzerepelBenne
  WHERE év = 1996 AND nem = 'N' AND színészNév = név;
```

Ez azt jelenti, hogy összekapcsoljuk a `FilmSzínész` és a `SzerepelBenne` relációkat, felhasználva azt a feltételt, hogy a színész neve mindkét relációban azonos. Ezután kiválasztjuk azokat a sorokat, amelyben a megjelenés éve 1996, és a színész neme nő.

A fentihez hasonló egyszerű lekérdezést a lekérdezésfordító egy olyan logikai lekérdezéstervvé alakít, amelynek első lépése a `FROM` utáni relációk egyesítése a szorzat operátor felhasználásával. A következő lépés a `WHERE` záradéknak megfelelő kiválasztás végrehajtása, és az utolsó lépés ennek levetítése a `SELECT` záradékban szereplő listára. A fenti lekérdezésnek megfelelő algebrai kifejezést az alábbi ábra mutatja:



Több olyan kifejezés is létezik, amelyik ekvivalens az ábrán látható kifejezéssel abban az értelemben, hogy ugyanazokra a relációkra ezen kifejezések eredménye ugyanaz lesz. A következő ábrán egy ilyen ekvivalens kifejezésre láthatunk példát:



Ez a kifejezés jelentősen eltér az előző tervtől. Először is észrevehetjük, hogy a `WHERE` záradék `színészNév = név` feltételét ebben az esetben a szorzatra alkalmazzuk, amely ily módon egy egyenlőségen alapuló összekapcsolássá válik. Itt tehát egy kiválasztás és egy szorzat összekapcsolássá történő átalakítását alkalmaztuk. Az összekapcsolások általában kevesebb sort eredményeznek, éppen ezért, ha lehet választani, akkor inkább az összekapcsolást választjuk a szorzat helyett.

Másodszor azt vehetjük észre, hogy a `WHERE` záradékban szereplő két feltételt szétszedtük két σ műveletté, és ezeket a műveleteket „lejjebb csúsztattuk" a fában, egészen a megfelelő relációkig. A $\sigma_{\text{év} = 1996}$ kiválasztást például közvetlenül a `SzerepelBenne` relációra alkalmazzuk, mivel ez az egyetlen olyan reláció, amely `év` attribútumot visz az előző terv szorzatába. Általános szabály, hogy érdemes a kiválasztást (rendszerint) a lehető leghamarabb elvégezni. Mivel a szorzások és az

összekapcsolások jellegzetesen több időt vesznek igénybe, mint a kiválasztások, ezért a relációk méretének mihamarabbi csökkentése sokkal jobban lecsökkenti az összekapcsoláshoz szükséges időt, mint amennyire megnöveli a kiválasztáshoz szükséges időt. A relációk méretének mihamarabbi csökkentését pedig úgy érhetjük el, hogy a kiválasztásokat minél lejjebb csúsztatjuk a fában úgy, ahogyan az a fenti ábrán látható.

Algebrai szabályok lekérdezéstervek javítására

Miután az elemző és az előfeldolgozó elvégezte a feladatát, a logikai lekérdezéstervet generáló az elemzőfát egy kifejezéssé transzformálja, amely nagyrészt vagy teljesen a relációalgebra operátoraiból áll. Ezután a relációalgebrára érvényes algebrai szabályok felhasználásával megpróbálunk javítani a lekérdezés algebrai kifejezésén. Most összegyűjtjük azokat az algebrai szabályokat, amelyek egy kifejezésfát olyan ekvivalens kifejezésfává alakítanak, amelyhez esetleg hatékonyabb fizikai lekérdezésterv tartozik.

Az ilyen algebrai transzformációk alkalmazásának eredménye a logikai lekérdezésterv, ami egyben a lekérdezésátírási fázis kimenete. Ezután történik a logikai lekérdezésterv átfordítása fizikai lekérdezéstervvé, amikor is az optimalizáló számos döntést hoz az operátorok megvalósításával kapcsolatban. Egy másik (a gyakorlatban nem nagyon használt) lehetőség az, hogy több jó logikai tervet generálunk a lekérdezésátírási fázisban, és az ezekből generált fizikai terveket vizsgáljuk meg, végül a legjobb fizikai tervet kiválasztjuk.

Kommutatív és asszociatív szabályok

A különféle kifejezések egyszerűsítésére használt legáltalánosabb szabályok a kommutatív és asszociatív szabályok. Egy operátorra vonatkozó *kommutatív szabály* azt mondja ki, hogy nem számít, hogy milyen sorrendben adjuk meg az operátor argumentumait, az eredmény ugyanaz lesz. Egy operátorra vonatkozó *asszociatív szabály* azt mondja ki, hogy ha az operátort egymás után kétszer használjuk, akkor egyaránt csoportosíthatunk balról vagy jobbról. Az aritmetikában például az összeadás és a szorzás kommutatív és asszociatív műveletek, míg a kivonás se nem kommutatív, se nem asszociatív. Amikor egy operátor egyszerre kommutatív és asszociatív is, akkor ha bármennyi operandust kötünk is össze az operátorral, az operandusokat tetszés szerint átrendezhetjük anélkül, hogy az eredmény megváltozna. Például: $((w + x) + y) + z = (y + x) + (z + w)$.

A relációalgebra következő operátorai egyszerre kommutatívak és asszociatívak:

- $R \times S = S \times R$; $(R \times S) \times T = R \times (S \times T)$
- $R * S = S * R$; $(R * S) * T = R * (S * T)$
- $R \cup S = S \cup R$; $(R \cup S) \cup T = R \cup (S \cup T)$
- $R \cap S = S \cap R$; $(R \cap S) \cap T = R \cap (S \cap T)$

Az egyesítésre és a metszetre vonatkozó szabályok egyaránt érvényesek a halmazokra és a multihalmazokra vonatkozó műveletek ($\cup_S$, $\cup_B$, $\cap_S$, $\cap_B$) esetén is (az első két művelet hatása ugyanaz halmazok és multihalmazok esetén).

A szabályokat nem bizonyítjuk, csak a bizonyítás általános menetét adjuk meg: be kell látni, hogy a bal oldali kifejezés által előállított minden sort a jobb oldali kifejezés is előállítja (mégpedig pontosan ugyanannyiszor), valamint fordítva. Fontos figyelembe venni, hogy a relációalgebrában az attribútumok sorrendje nem kötött, az első két kommutatív szabály bizonyítása ezen alapul.

A fenti felsorolásban nem szerepel a théta-összekapcsolás. Ez az operátor kommutatív, és ha az érintett feltételeknek van értelme ott, ahová kerülnek, akkor asszociatív is. Vannak azonban olyan esetek, amikor

az asszociatív szabály nem alkalmazható, mert a feltételek nem az éppen összekapcsolni kívánt relációk attribútumaira vonatkoznak.

Példa. Tegyük fel, hogy van három relációnk: `R(a,b)`, `S(b,c)` és `T(c,d)`. Ha érvényes lenne az asszociatív szabály, akkor a következő egyenlőség teljesülne:

$(R \otimes_{R.b > S.b} S) \otimes_{a < d} T = R \otimes_{R.b > S.b} (S \otimes_{a < d} T)$

`S`-et és `T`-t azonban nem kapcsolhatjuk össze az `a < d` feltétel alapján, hiszen `a` sem `S`-nek, sem `T`-nek nem attribútuma.

Óvatosnak kell lennünk, amikor a halmazokra vonatkozó szabályokat multihalmazokra próbáljuk alkalmazni. A metszet disztributív szabálya például érvényes halmazokra, de nem érvényes multihalmazokra:

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Tegyük fel például, hogy `A`, `B` és `C` mindegyike az `{x}` multihalmaz. Ekkor $B \cup_B C = \{x,x\}$ és $A \cap_B (B \cup_B C) = \{x\}$, hiszen a multihalmazokra vonatkozó metszet az előfordulások számának minimumát veszi. Ugyanakkor $A \cap_B B$ és $A \cap_B C$ mindegyike `{x}`, vagyis a jobb oldali kifejezés: $(A \cap_B B) \cup_B (A \cap_B C) = \{x,x\}$, ami különbözik a bal oldalon kapott `{x}`-től.

Kiválasztással kapcsolatos szabályok

A kiválasztás művelete döntő jelentőségű a lekérdezésoptimalizálás szempontjából. Mivel a kiválasztások lényegesen csökkenthetik a relációk méretét, a hatékony lekérdezésfeldolgozás egyik legfontosabb szabálya, hogy a kiválasztásokat vigyük lefelé a fában mindaddig, amíg ez nem változtatja meg a kifejezés eredményét. A korai lekérdezésoptimalizálók valóban ennek a transzformációnak a változatait használták a jó logikai tervhez vezető elsődleges stratégiaként. Amint röviden rá fogunk mutatni, a „kiválasztások tologatása lefelé a fában” transzformáció már nem teljesen általános, de a „kiválasztások tologatása” elv a lekérdezésoptimalizálóknak még mindig egy jelentős eszköze.

Az első szabályok azt mondják, hogy ha egy kiválasztás feltétele összetett (azaz `AND` vagy `OR` által összekapcsolt részfeltételekből áll), akkor azzal segíthetünk, hogy a feltételt szétvágjuk az alkotóelemeire. Ezt az indokolja, hogy egy olyan kiválasztás, amelynek feltétele kevesebb attribútumot tartalmaz, mint az egész feltétel, esetleg elmozdítható egy megfelelő helyre, ahová az eredeti kiválasztás nem. Az első két szabály, amelyeket *szétvágási szabályoknak* hívunk, a következő:

- $\sigma_{C1 \text{ AND } C2}(R) = \sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1}(R) \cap_B \sigma_{C2}(R)$
- $\sigma_{C1 \text{ OR } C2}(R) = \sigma_{C1}(R) \cup_S \sigma_{C2}(R)$, ha `R` halmaz.

Az `OR`-ra vonatkozó szabály tehát csak akkor működik, ha az `R` reláció halmaz. Ha ugyanis `R` multihalmaz lenne, akkor a halmazegyesítés eltüntetné az ismétlődéseket.

Mivel az `AND` és az `OR` műveletek kommutatívak, C_1 és C_2 sorrendje nem kötött. Általánosabban azt mondhatjuk, hogy a σ operátor tetszőleges sorozata esetén a sorrend felcserélhető:

- $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$

Példa. Legyen `R(a,b,c)` egy reláció. Ekkor a $\sigma_{(a = 1 \text{ OR } a = 3) \text{ AND } b < c}(R)$ kifejezés szétvágható az alábbivá: $\sigma_{a = 1 \text{ OR } a = 3}(\sigma_{b < c}(R))$. Ez a kifejezés az `OR` műveletnél tovább vágható a következővé: $\sigma_{a = 1}(\sigma_{b < c}(R)) \cup_B \sigma_{a = 3}(\sigma_{b < c}(R))$. Mivel esetünkben lehetetlen, hogy egy sorra mind az `a = 1`, mind az `a = 3` teljesüljön, ez az átalakítás érvényes, függetlenül attól, hogy `R` halmaz-e vagy sem. Az $\cup_S$ művelet viszont nem alkalmazható, ha megengedjük, hogy `R` multihalmaz legyen. Összefoglalva tehát: ha

az OR művelet két oldalán egymást kizáró feltételek állnak, akkor megengedhetjük, hogy R multihalmaz legyen, de ekkor nem az $\cup_S$, hanem az $\cup_B$ műveletet kell alkalmaznunk:

- $\sigma_{C1\ OR\ C2}(R) = \sigma_{C1}(R) \cup_B \sigma_{C2}(R)$, ha C_1 és C_2 kizárják egymást.

A szétvágást úgy is elkezdhettük volna, hogy a $\sigma_{b<c}$-ből készítünk külső műveletet: $\sigma_{b<c}(\sigma_{a=1\ OR\ a=3}(R))$. Ebből az OR szétvágása után azt kapjuk, hogy $\sigma_{b<c}(\sigma_{a=1}(R) \cup_B \sigma_{a=3}(R))$, ami az első kifejezéssel ekvivalens, de attól valamelyest különböző.

A kiválasztásra vonatkozó szabályok következő csoportja azt teszi lehetővé, hogy a kiválasztásokat áttoljuk az egyesítés, metszet, különbség, szorzat és összekapcsolás operátorokon. Háromféle szabály van, attól függően, hogy kötelező vagy opcionális bevinni a kiválasztást az egyes argumentumokhoz:

1. Egyesítés esetén a kiválasztást mindkét argumentumra alkalmazni kell.
2. Különbség esetén a kiválasztást az első argumentumra alkalmazni kell, a másodikra pedig lehet.
3. A többi operátor esetében csak azt követeljük meg, hogy a kiválasztást egy argumentumra alkalmazzuk. A szorzatnál és az összekapcsolásnál lehet, hogy nincs annak értelme, hogy a kiválasztást mindkét argumentumhoz bevigyük, mivel egy argumentum vagy rendelkezik azokkal az attribútumokkal, amelyeket a kiválasztás megkíván, vagy nem. Ha lehetséges is a mindkettőre történő alkalmazás, akkor sem biztos, hogy ez javít a terven.

Az egyesítésre vonatkozó szabály:

- $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$

Itt kötelezően le kell vinni a kiválasztást a fa mindkét ágán.

A különbségre vonatkozó szabály két változata:

- $\sigma_C(R - S) = \sigma_C(R) - S$
- $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$

Az első argumentumra tehát kötelező alkalmazni a kiválasztást, a másodikra pedig alkalmazhatjuk.

A következő szabályok megengedik, hogy a kiválasztást az egyik vagy mindkét argumentumhoz bevigyük. Ha a kiválasztás σ_C, akkor ezt a kiválasztást csak olyan relációhoz tolhatjuk, amely rendelkezik a C-ben szereplő összes attribútummal. Az alábbi szabályok feltételezik, hogy az R relációban megvan az összes C-ben szereplő attribútum:

- $\sigma_C(R \times S) = \sigma_C(R) \times S$
- $\sigma_C(R * S) = \sigma_C(R) * S$
- $\sigma_C(R \otimes_D S) = \sigma_C(R) \otimes_D S$
- $\sigma_C(R \cap S) = \sigma_C(R) \cap S$

Ha C-ben csak S-beli attribútumok szerepelnek, akkor a fenti szabályok a következőképpen módosulnak:

- $\sigma_C(R \times S) = R \times \sigma_C(S)$
- $\sigma_C(R * S) = R * \sigma_C(S)$
- $\sigma_C(R \otimes_D S) = R \otimes_D \sigma_C(S)$
- $\sigma_C(R \cap S) = R \cap \sigma_C(S)$

Ha az R és S relációk mindegyikében szerepel az összes C-beli attribútum, akkor az alábbi szabályok is használhatók:

- $\sigma_C(R * S) = \sigma_C(R) * \sigma_C(S)$
- $\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$

Nem alkalmazható ilyen szabály a szorzatra és a théta-összekapcsolásra, ezekben az esetekben ugyanis R-nek és S-nek nincsenek közös attribútumai. A metszet esetében viszont a szabály mindig érvényes lesz, hiszen ekkor R és S sémája ugyanaz kell, hogy legyen.

Példa. Tekintsük az R(a,b) és S(b,c) relációkat és a következő kifejezést:

$\sigma_{(a = 1 \text{ OR } a = 3) \text{ AND } b < c}(R * S)$

A b < c feltétel egyedül S-re alkalmazható, az a = 1 OR a = 3 feltétel pedig csak R-re. Ezért a két részfeltételt összekötő AND szétvágásával kezdjük:

$\sigma_{a = 1 \text{ OR } a = 3}(\sigma_{b < c}(R * S))$

Ezt követően bevihetjük a $\sigma_{b < c}$ kiválasztást S-hez, ami az alábbi kifejezést adja:

$\sigma_{a = 1 \text{ OR } a = 3}(R * \sigma_{b < c}(S))$

Végül az első feltételt bevisszük R-hez:

$\sigma_{a = 1 \text{ OR } a = 3}(R) * \sigma_{b < c}(S)$

Ha akarjuk, szétvághatjuk az OR-ral kapcsolt két részfeltételt is, ez azonban vagy előnyös, vagy nem.

A kiválasztással kapcsolatban természetesen még számos egyéb szabályt megfogalmazhatnánk. Ilyenek például azok a szabályok, amelyek olyan speciális esetekkel foglalkoznak, mint például amikor egy reláció üres, egy feltétel azonosan igaz vagy hamis, vagy a teljes attribútumlistára történik vetítés. Íme három példa a speciális esetekre vonatkozó szabályok közül:

- $\sigma_C(R) = \emptyset$, ha $R = \emptyset$, azaz üres relációra vonatkozó bármilyen kiválasztás üres relációt ad.
- $\sigma_C(R) = R$, ha a C feltétel mindig igaz. (Vigyázzunk, hogy például az x > 10 OR x ≤ 10 feltétel csak akkor azonosan igaz, ha x nem vehet fel NULL értéket!)
- $R \cup S = S$, ha $R = \emptyset$.

Kiválasztások tologatása

Amint már említettük, egy kiválasztás tologatása lefelé a fában (az előző részben szereplő valamely szabály bal oldalának helyettesítése annak jobb oldalával) a lekérdezésoptimalizáló egyik leghatékonyabb eszköze. Sokáig azt feltételezték, hogy úgy optimalizálhatunk, hogy a kiválasztásra vonatkozó szabályokat ebbe az irányba alkalmazzuk. Amikor azonban általánossá vált a nézettáblák támogatása, úgy találták, hogy bizonyos esetekben lényeges volt, hogy egy kiválasztást először olyan fentre vigyünk a fában, amennyire lehet, és utána tologassuk lefelé a kiválasztásokat a lehetséges ágakon. A kiválasztások tologatásának jó megközelítését egy példával szemléltetjük.

Példa. Vegyünk fel egy újabb relációt a korábbi kettő mellé:

```
Film(cím, év, hossz, stúdióNév)
```

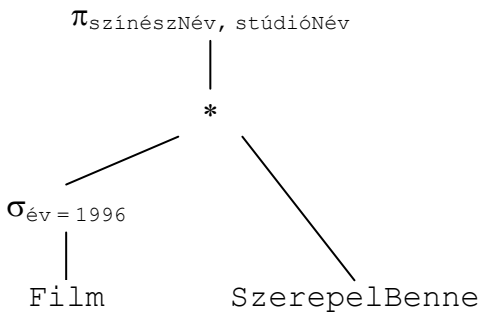
Legyen adott az alábbiakban definiált nézettábla:

```
CREATE VIEW Filmek1996 AS
  SELECT * FROM Film WHERE év = 1996;
```

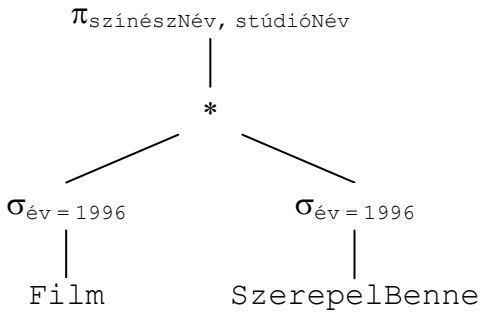
A „Mely színészek mely stúdiónak dolgoztak 1996-ban?” kérdést megfogalmazó SQL-lekérdezés:

```
SELECT színészNév, stúdióNév
  FROM Filmek1996 NATURAL JOIN SzerepelBenne;
```

A Filmek1996 nézettáblát a következő relációalgebrai kifejezés definiálja: $\sigma_{év = 1996}(Film)$. A fenti lekérdezéshez tehát a következő kifejezésfa (logikai lekérdezésterv) tartozik:



A kifejezésben szereplő egyetlen kiválasztás már olyan lent van a fában, amennyire csak lehet, így nincs lehetőség a kiválasztás tologatására lefelé a fában. A $\sigma_C(R * S) = \sigma_C(R) * S$ szabályt viszont alkalmazhatjuk visszafelé, hogy a $\sigma_{év = 1996}$ kiválasztást az összekapcsolás fölé vigyük. Ezután, mivel az `év` a `Film` és a `SzerepelBenne` relációknak egyaránt attribútuma, a kiválasztást az összekapcsoláshoz tartozó csomópont mindkét gyereke felé levihetjük. Az így kapott terv valószínűleg javulást jelent, hiszen a `SzerepelBenne` reláció méretét csökkentjük, mielőtt összekapcsoljuk az 1996-os filmekkel:



Vetítéssel kapcsolatos szabályok

A kiválasztáshoz hasonlóan a vetítéseket is tolhatjuk lefelé, más operátorokon keresztül. A vetítések tologatása abban különbözik a kiválasztások tologatásától, hogy amikor vetítést tolunk, akkor a vetítés általában ott is megmarad, ahol van. Úgy is mondhatjuk, hogy a vetítés „tolása” valójában egy új vetítés bevezetését jelenti valahol a létező vetítés alatt.

A vetítések tologatása hasznos ugyan, de általában nem annyira, mint a kiválasztás tologatása. Ennek az az oka, hogy míg a kiválasztás gyakran nagymértékben csökkenti egy reláció méretét, vetítés során a sorok száma ugyanaz marad, csak a sorok hossza csökken. Sőt, amint azt a vetítés definíciójánál láthattuk, a vetítés néha növeli a sorok hosszát.

A vetítésre vonatkozó szabályok mögött az alábbi alapelv húzódik meg:

- A kifejezéslistában bárhol bevezethetünk egy vetítést mindaddig, amíg az csakis olyan attribútumokat tüntet el, amelyeket egyetlen fentebb elhelyezkedő operátor sem használ, valamint a teljes kifejezés eredményében sem szerepelnek.

A szabályok legegyszerűbb alakjaiban a bevezetett vetítések mind egyszerűek:

- $\pi_L(R * S) = \pi_L(\pi_M(R) * \pi_N(S))$, ahol `M` az `R` reláció azon attribútumainak listája, amelyek vagy összekapcsolási attribútumok (azaz `R` és `S` sémájában egyaránt szerepelnek), vagy bemeneti attribútumai π_L-nek, `N` pedig `S` azon attribútumainak listája, amelyek vagy összekapcsolási attribútumok, vagy bemeneti attribútumai π_L-nek.

- $\pi_L(R \otimes_C S) = \pi_L(\pi_M(R) \otimes_C \pi_N(S))$, ahol M az R reláció azon attribútumainak listája, amelyek vagy összekapcsolási attribútumok (azaz előfordulnak a C feltételben), vagy bemeneti attribútumai π_L-nek, N pedig S azon attribútumainak listája, amelyek vagy összekapcsolási attribútumok, vagy bemeneti attribútumai π_L-nek.
- $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$, ahol M az R, N pedig az S reláció azon attribútumainak listája, amelyek bemeneti attribútumai π_L-nek.

Példa. Legyen $R(a,b,c)$ és $S(c,d,e)$ két reláció. Vegyük a következő kifejezést:

$\pi_{a+e \to x,\, b \to y}(R * S)$

A vetítés bemeneti attribútumai a, b és e, az egyetlen összekapcsolási attribútum c. A vetítések összekapcsolás alá történő eltolásának szabályát alkalmazva a következő ekvivalens kifejezést kapjuk:

$\pi_{a+e \to x,\, b \to y}(\pi_{a,\, b,\, c}(R) * \pi_{c,\, e}(S))$

Vegyük észre, hogy a $\pi_{a,\, b,\, c}(R)$ egy triviális vetítés, ami R összes attribútumára vetít. Ez a vetítés tehát kiküszöbölhető, ami egy harmadik ekvivalens kifejezést eredményez:

$\pi_{a+e \to x,\, b \to y}(R * \pi_{c,\, e}(S))$

Az egyetlen változás az eredetihez képest tehát az, hogy az összekapcsolás előtt S-ből elhagyjuk a d attribútumot.

Egy vetítést teljesen végrehajthatunk egy multihalmaz-egyesítés előtt:

- $\pi_L(R \cup_B S) = \pi_L(R) \cup_B \pi_L(S)$

Nem vihetők azonban a vetítések sem a halmazegyesítések, sem a metszet és a különbség egyik formája elé sem.

Példa. Legyen $R(a,b)$ az $\{(1, 2)\}$, $S(a,b)$ pedig az $\{(1, 3)\}$ reláció. Ekkor $\pi_a(R \cap S) = \pi_a(\varnothing) = \varnothing$, $\pi_a(R) \cap \pi_a(S) = \{(1)\} \cap \{(1)\} = \{(1)\}$.

Ha a vetítés számításokat tartalmaz, és a vetítési lista valamely kifejezéséhez tartozó bemeneti attribútumok teljes egészében egy, a vetítés alatt elhelyezkedő összekapcsolás vagy szorzat egyik argumentumához tartoznak, akkor megvan az a lehetőségünk, hogy az adott számítást közvetlenül azon az argumentumon végezzük el. Például:

- $\pi_{L1,\, E \to x,\, L2}(R * S) = \pi_{L1,\, x,\, L2}(\pi_{M,\, E \to x}(R) * \pi_N(S))$, ha az E kifejezésben szereplő attribútumok az R reláció attribútumai.

Példa. Legyen ismét $R(a,b,c)$ és $S(c,d,e)$ két reláció, és tekintsük a következő kifejezést:

$\pi_{a+b \to x,\, d+e \to y}(R * S)$

Az $a + b$ összeadást és annak x-re történő átnevezését közvetlenül az R relációhoz vihetjük, és ugyanezt tehetjük a $d + e$ összeggel S vonatkozásában. Az így kapott ekvivalens kifejezés:

$\pi_{x,\, y}(\pi_{a+b \to x,\, c}(R) * \pi_{d+e \to y,\, c}(S))$

Speciálisan kell kezelni azt az esetet, ha x vagy y megegyezik c-vel. Ekkor nem nevezhetnénk át az összeget c-re, mert egy relációnak nem lehet két különböző attribútuma ugyanazzal a névvel. Be kellene vezetni egy ideiglenes nevet, és az összekapcsolás fölött végre kellene hajtani egy további átnevezést. A $\pi_{a+b \to c,\, d+e \to y}(R * S)$ kifejezés például a következő kifejezéssé alakítható át:

$\pi_{z \to c,\, y}(\pi_{a+b \to z,\, c}(R) * \pi_{d+e \to y,\, c}(S))$

Egy vetítést be lehet iktatni egy kiválasztás alá is:

- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$, ahol M azoknak az attribútumoknak a listája, amelyek vagy bemeneti attribútumai π_L-nek, vagy szerepelnek a C feltételben.

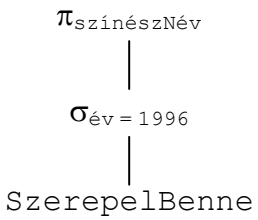
Azaz ahogy az előző példában, most is lehetséges, hogy az L lista számításait inkább M-ben végezzük el, feltéve, hogy a C feltétel nem igényli L azon attribútumait, amelyek szerepelnek valamelyik számításban.

Gyakran akkor is lejjebb akarjuk vinni a vetítéseket a kifejezésfában, ha fent ott kell hagyni egy másik vetítést, mert a vetítések általában csökkentik a sorok méretét, és így egy köztes reláció által elfoglalt blokkok számát. Vigyáznunk kell azonban, amikor ezt tesszük, mert vannak tipikus példák, amikor egy vetítés levitele időbe kerül.

Példa. Vegyük azt a lekérdezést, ami az 1996-ban dolgozó színészeket keresi:

```
SELECT színészNév FROM SzerepelBenne WHERE év = 1996;
```

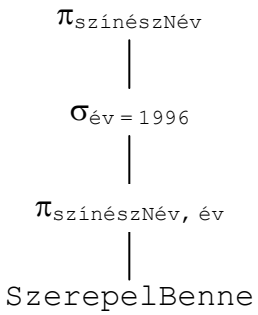
Az alábbi ábra mutatja ennek a lekérdezésnek a közvetlen átalakítását logikai lekérdezéstervvé:



A kiválasztás alá beilleszthetünk egy vetítést két attribútummal:

1. színészNév, ugyanis ez az attribútum kell az eredményhez;
2. év, mert ez az attribútum szükséges a kiválasztási feltételhez.

Az eredmény a következő ábrán látható:



Ha a SzerepelBenne nem tárolt reláció lenne, hanem valamilyen művelet által létrehozott reláció, akkor lenne értelme ennek a tervnek. „Futószalagosíthatjuk" a vetítést (lásd később), amint a művelet eredményét képező reláció sorait előállítjuk, egyszerűen elhagyva a nem használt filmCím attribútumot.

A mi esetünkben azonban a SzerepelBenne egy tárolt reláció. Az alsó vetítés valójában nagy időpocsékolást jelent, különösen akkor, ha létezik index az év attribútumra. Ekkor az eredeti lekérdezésterven alapuló fizikai lekérdezésterv először az indexet használná az olyan sorok megtalálására, ahol az év 1996, ami feltehetően a soroknak csak egy kis hányadát jelenti. Ha a vetítést végezzük el először, akkor a SzerepelBenne reláció minden sorát be kell olvasni, és vetíteni kell.

Hogy a dolgok még rosszabbul nézzenek ki, az év-hez tartozó index valószínűleg nem használható a vetített $\pi_{\text{színészNév, év}}(\text{SzerepelBenne})$ relációhoz, így a kiválasztásnak a vetítés eredményeként megkapott összes sort végig kell olvasnia.

Szorzatra és összekapcsolásra vonatkozó szabályok

Korábban már láttuk a szorzat és az összekapcsolás kommutatív és asszociatív szabályait. Van néhány további szabály, amelyek közvetlenül az összekapcsolás definíciójából következnek:

- $R * S = \pi_L(\sigma_C(R \times S))$
- $R \otimes_C S = \sigma_C(R \times S)$

Ezeknek a szabályoknak a használatát a definíciónál adott példák szemléltették. A gyakorlatban rendszerint jobbról balra alkalmazzuk ezeket a szabályokat, vagyis egy szorzatot követő kiválasztást azonosítunk összekapcsolásként. Ennek az az oka, hogy az összekapcsolások kiszámításához használt algoritmusok általában sokkal gyorsabbak, mint az olyan algoritmusok, amelyek egy szorzatot és egy, a szorzat (nagyon nagy méretű) eredményére alkalmazott kiválasztást számítanak ki.

Ismétlődések elhagyására vonatkozó szabályok

Az ismétlődéseket eltávolító δ operátort sok operátoron keresztül lehet tolni, de nem mindegyiken. A δ lefelé történő mozgatása a fában csökkenti a köztes relációk méretét, így kifizetődő lehet. Sőt a δ néha olyan helyre vihető, ahol egyszerűen elhagyható, mert olyan relációra vonatkozik, amelyről tudni lehet, hogy nem tartalmaz ismétlődéseket:

- $\delta(R) = R$, ha R-ben nincsenek ismétlődések. Ilyen eset például, ha R
  a) egy tárolt reláció, amelyhez elsődleges kulcsot deklaráltunk;
  b) egy γ művelet eredményeként kapott reláció, mivel egy csoportosítás eredménye egy ismétlődések nélküli reláció;
  c) egy halmazművelet eredménye.

Az alábbi néhány szabály a δ operátort más operátorokon tolja keresztül:

- $\delta(R \times S) = \delta(R) \times \delta(S)$
- $\delta(R * S) = \delta(R) * \delta(S)$
- $\delta(R \otimes_C S) = \delta(R) \otimes_C \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$

A δ odavihető egy metszet egyik vagy mindkét argumentumához is:

- $\delta(R \cap_B S) = \delta(R) \cap_B S = R \cap_B \delta(S) = \delta(R) \cap_B \delta(S)$

Ugyanakkor viszont a δ általában nem vihető át az $\cup_B$, $-_B$ vagy π operátorokon.

Példa. Legyen az R reláció olyan, amelyben a t sor két példányban szerepel, az S pedig olyan, amelyben a t sor egy példányban szerepel. Ekkor a $\delta(R \cup_B S)$ egy példányát, míg a $\delta(R) \cup_B \delta(S)$ két példányát tartalmazza a t sornak. A $\delta(R -_B S)$ egy példányát tartalmazza a t sornak, míg a $\delta(R) -_B \delta(S)$ nem tartalmazza a t sort.

Vegyük most azt a $T(a,b)$ relációt, amely az (1, 2) és (1, 3) sorok egy-egy példányát tartalmazza és mást nem. Ekkor a $\delta(\pi_a(T))$ eredményében az (1) sor egyszer szerepel, míg a $\pi_a(\delta(T))$ eredményében az (1) sor kétszer fordul elő.

A δ felcserélésének az $\cup_S$, $\cap_S$ és $-_S$ operátorokkal nincs értelme. Ehelyett a δ elhagyható, mivel ezen halmazműveletek megvalósítása magában foglalja az ismétlődések eltüntetésének folyamatát, ami egyenértékű a δ alkalmazásával. Például:

- $\delta(R \cup_S S) = R \cup_S S$

Csoportosításra és összesítésre vonatkozó szabályok

Ha megnézzük a γ operátort, azt látjuk, hogy sok transzformáció alkalmazhatósága a használt összesítő operátor részleteitől függ. Emiatt nem állíthatunk fel szabályokat olyan általánosságban, mint ahogyan a többi operátor esetében tettük. Kivételt képez az az eset, amikor egy γ elnyel egy δ-t:

- $\delta(\gamma_L(R)) = \gamma_L(R)$

Egy másik általános szabály az, hogy ha úgy kívánjuk, akkor a γ operátor alkalmazása előtt az argumentumban nem használt attribútumokat elhagyhatjuk egy vetítés segítségével:

- $\gamma_L(R) = \gamma_L(\pi_M(R))$, ahol M az R reláció azon attribútumainak listája, amelyek előfordulnak L-ben.

Annak oka, hogy más transzformációk a γ-ban szereplő összesítésektől függnek, a következő: Bizonyos összesítéseket (MIN és MAX) nem befolyásol az ismétlődések jelenléte vagy hiánya, a többi összesítés (SUM, COUNT és AVG) viszont általában más értéket produkál, ha az összesítés alkalmazása előtt megszüntetjük az ismétlődéseket.

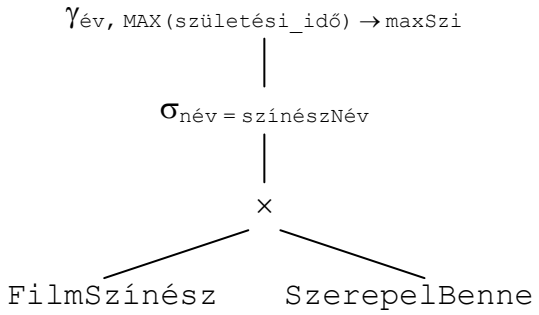
Egy γ_L operátort *ismétlődésérzéketlennek* (duplicate insensitive) nevezünk, ha L-ben csak ismétlődésérzéketlen összesítő operátorok (MIN és/vagy MAX) szerepelnek. Ezek után:

- $\gamma_L(R) = \gamma_L(\delta(R))$, ha γ_L ismétlődésérzéketlen.

Példa. Tegyük fel, hogy minden évhez meg akarjuk keresni az adott évben valamilyen filmben szereplő legfiatalabb színész születési idejét. Ez az alábbi lekérdezéssel fejezhető ki:

```
SELECT év, MAX(születési_idő) FROM FilmSzínész, SzerepelBenne
  WHERE név = színészNév GROUP BY év;
```

A közvetlenül a lekérdezésből kapott kiindulási logikai lekérdezéstervet az alábbi ábrán láthatjuk:

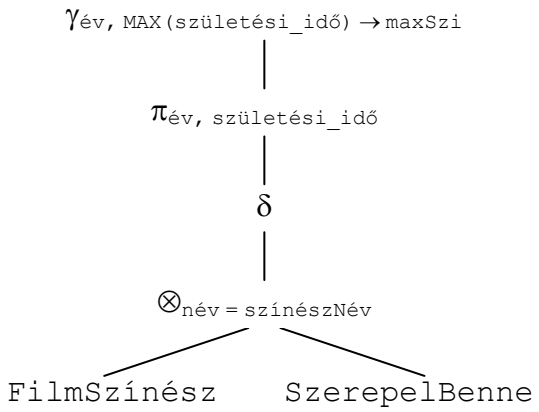


A from-listát egy szorzat, a WHERE záradékot egy efölött lévő kiválasztás, a csoportosítást és az összesítést pedig egy ezek fölött elhelyezkedő γ operátor fejez ki.

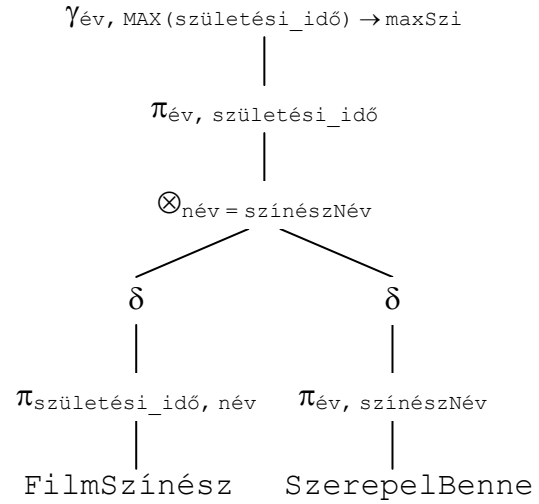
Ezen a terven több átalakítás is elvégezhető:

1. a kiválasztás és a szorzat összevonása egy egyenlőségen alapuló összekapcsolássá;
2. egy δ beillesztése a γ alá, mivel γ ismétlődésérzéketlen;
3. egy olyan π vetítés beillesztése a γ és az újonnan bevezetett δ közé, ami az év-re és a születési_idő-re, vagyis a γ szempontjából lényeges attribútumokra terjed ki.

Az eredményül kapott terv a következő:



Ezután levihetjük a δ-t a ⊗ alá, ez alá pedig π-ket vezethetünk be, ha úgy kívánjuk. Az így kapott lekérdezéstervet mutatja a következő ábra:



Ha a `név` kulcsa a `FilmSzínész` relációnak, akkor az ehhez a relációhoz vezető ágon a δ elhagyható.

Elemzőfák átalakítása logikai lekérdezéstervekké

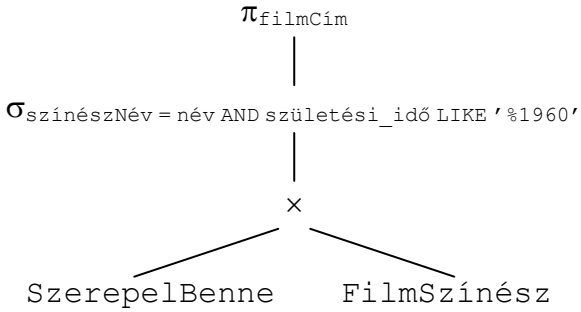
Az algebrai szabályok ismertetése után térjünk vissza a lekérdezésfordító tárgyalásához. Miután létrehoztunk egy elemzőfát, a következő feladat az elemzőfa átalakítása a jónak vélt logikai lekérdezéstervvé. Ezt két lépésben hajtjuk végre. Az első lépésben az elemzőfa csomópontjait és struktúráit helyettesítjük (megfelelő csoportosításban) a relációalgebra operátoraival. A második lépésben vesszük az így előállított relációalgebrai kifejezést, és egy olyan kifejezéssé alakítjuk, amely várhatóan a leghatékonyabb fizikai lekérdezéstervvé konvertálható.

Átfordítás relációalgebrába

Most formalizmusok nélkül bevezetünk néhány olyan szabályt, amely egy SQL-elemzőfa algebrai logikai lekérdezéstervvé történő transzformálásával kapcsolatos. Az első – talán legfontosabb – szabály lehetővé teszi számunkra, hogy minden „egyszerű” `SELECT…FROM…WHERE` szerkezetet közvetlenül konvertáljunk a relációalgebrába:

- Ha adott egy <Lekérdezés>, ami egy <SFW> struktúra, és a <Feltétel> ebben a struktúrában nem tartalmaz alkérdést, akkor a teljes struktúra (a select-lista, a from-lista és a feltétel) egy olyan relációalgebrai kifejezéssel helyettesíthető, amely alulról felfelé az alábbiakból áll:
  1. a <FromLista>-ban szereplő relációk szorzata, ha a lista több relációt tartalmaz, illetve maga a reláció, ha a lista egyelemű;
  2. egy σ_C kiválasztás, ahol C a helyettesítés alatt álló struktúra <Feltétel> kifejezése;
  3. egy π_L vetítés, ahol L a <SelLista> attribútumlistája.

Példa. Tekintsük az elemzésről szóló részben tárgyalt második példában látható elemzőfát (6. oldal). A fenti szabály a teljes elemzőfára alkalmazható. Vesszük a from-lista két relációjának szorzatát, kiválasztunk a <Feltétel>-ben gyökerező részfának megfelelő feltétel alapján, és vetítünk a select-listára:



Ugyanez a transzformáció nem alkalmazható az első példa külső szintű lekérdezésénél (5. oldal), mert a feltétel alkérdést tartalmaz. (Az alkérdéseket tartalmazó feltételek kezelését a következő fejezetben fogjuk tárgyalni.) Alkalmazhatjuk viszont a szabályt ugyanezen példa alkérdésére, amiből a következő relációalgebrai kifejezést kapjuk:

$\pi_{\text{név}}(\sigma_{\text{születési\_idő LIKE '\%1960'}}(\text{FilmSzínész}))$

Elcsodálkozhatunk azon, hogy miért nem engedjük meg, hogy egy σ_C kiválasztás C feltétele alkérdést tartalmazzon. A relációalgebrában az a szokás, hogy egy művelet argumentumai (a nem indexként szereplő elemek) olyan kifejezések, amelyek relációkat eredményeznek. Másrészt viszont a paraméterek (az indexben megjelenő elemek) nem relációk. A σ_C-ben például a C paraméter egy logikai típusú kifejezés, a π_L-ben pedig az L paraméter egy attribútumokból vagy formulákból álló lista.

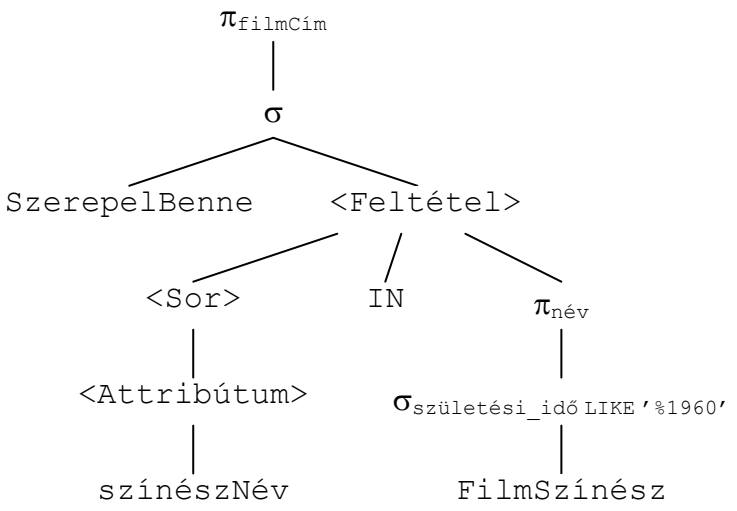
Ha követjük ezt a hagyományt, akkor egy paraméter alkalmazható az argumentum relációk minden egyes sorára, bármilyen számítást jelent is ez. A paraméterek használatára vonatkozó korlátozás egyszerűbbé teszi a lekérdezésoptimalizálást. Tegyük fel, hogy egy $\sigma_C(R)$ operátor C feltétele tartalmaz egy alkérdést. Ekkor C alkalmazása az R reláció egyes soraira igényli az alkérdés kiszámítását. A kérdés, hogy kiszámítjuk újra és újra R minden egyes soránál? Ez szükségtelenül drága volna, kivéve, ha az alkérdés *korrelatív*, azaz az értékei függnek valamitől, ami rajta kívül definiált. A legtöbb esetben még a korrelatív alkérdéseket is ki lehet értékelni anélkül, hogy azt minden sornál újra ki kellene számítani, feltéve, hogy a kiszámítást helyesen szervezzük meg.

Alkérdések eltávolítása feltételekből

Azokhoz az elemzőfákhoz, amelyekben van alkérdést tartalmazó <Feltétel>, bevezetünk egy közbeeső operátort, amely az elemzőfa szintaktikus kategóriái és a relációalgebrai operátorok között helyezkedik el, és relációkra vonatkozik. Ezt az operátort *kétargumentumú kiválasztásnak* nevezzük. A kétargumentumú kiválasztást a transzformált elemzőfában egy olyan csomópont képviseli, amelynek címkéje σ, mégpedig paraméterek nélkül. E csomópont alatt elhelyezkedik egy bal oldali gyerek, amely

azt az R relációt képviseli, amelyre a kiválasztás vonatkozik, valamint egy jobb oldali gyerek, ami az R soraira vonatkozó feltételt megtestesítő kifejezés. Mindkét argumentum ábrázolható mint elemzőfa, mint kifejezésfa és mint a kettő keveréke.

Példa. A következő ábrán láthatjuk az elemzésről szóló rész első példájában megadott elemzőfának (5. oldal) egy olyan átírását, amelyben kétargumentumú kiválasztás szerepel:



Több transzformációt hajtottunk végre, mire eljutottunk ehhez az ábrához:

1. Az alkérdést egy relációalgebrai kifejezéssel helyettesítettük az előző példa végén mondottak alapján.
2. A SELECT…FROM…WHERE kifejezésekhez bevezetett szabálynak megfelelően helyettesítettük a külső szintű lekérdezést; a szükséges kiválasztást azonban egy kétargumentumú kiválasztás segítségével fejeztük ki, és nem a relációalgebra hagyományos σ operátorával. Következésképpen az elemzőfa felső <Feltétel> csomópontját nem helyettesítettük, az megmaradt mint a kiválasztás egyik argumentuma, de a hozzá tartozó kifejezés egy részét helyettesítettük relációalgebrával.

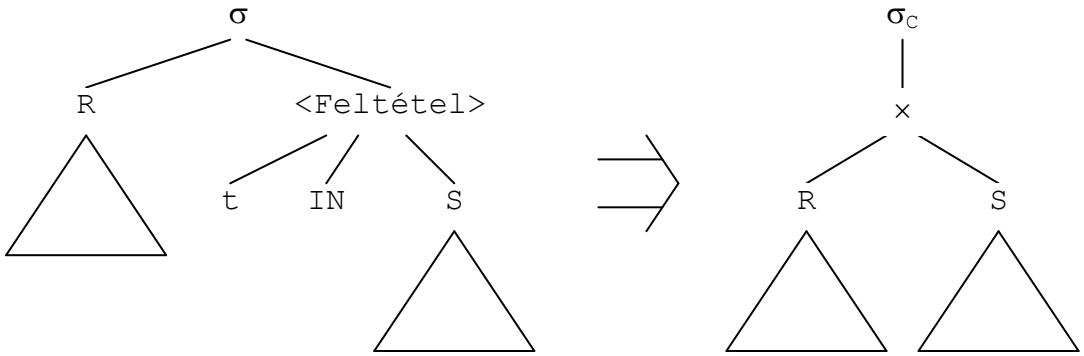
Ez a fa további transzformációt igényel, ezt tárgyaljuk következőként.

Szükségünk van olyan szabályokra, amelyek lehetővé teszik, hogy egy kétargumentumú kiválasztást egy relációalgebrai kiválasztással és egy másik relációalgebrai operátorral helyettesítsünk. A feltételek különböző formái külön szabályokat igényelhetnek. A legtöbb helyzetben a kétargumentumú kiválasztás eltávolítható, és tiszta relációalgebrai kifejezéshez juthatunk. Vannak azonban különleges esetek, amikor a kétargumentumú kiválasztást a helyén hagyjuk, és a logikai lekérdezésterv részének tekintjük.

Példaként megadjuk azt a szabályt, amelynek segítségével a fenti ábrán szereplő, IN operátort tartalmazó feltételt kezelhetjük. Vegyük észre, hogy az alkérdés a feltételben független (nem korrelatív), azaz a neki megfelelő reláció nem függ az éppen vizsgált sortól (tehát elég egyszer kiszámítani). Az ilyen feltételeket elimináló szabály informálisan így fogalmazható meg:

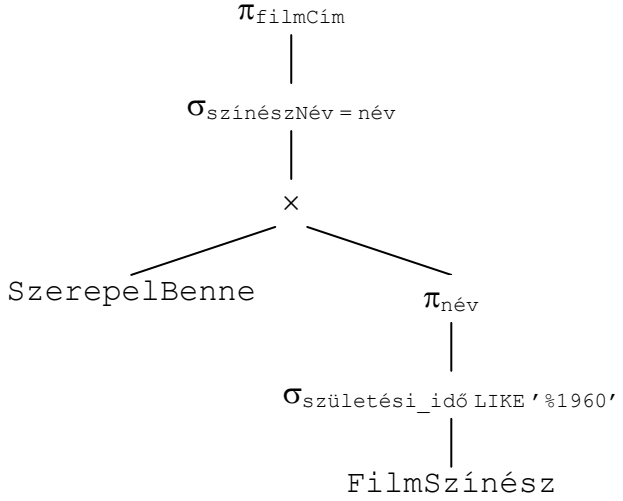
- Tegyük fel, hogy van egy kétargumentumú kiválasztás, amelynek első argumentuma egy R relációt képvisel, a második argumentuma pedig egy t IN S alakú kifejezés, ahol S egy független alkérdés, t pedig az R reláció bizonyos attribútumaiból összeállított sor. A fa az alábbi módon transzformálható:
  a) Helyettesítsük a <Feltétel>-t azzal a fával, ami nem más, mint az S kifejezés. Ha S-ben lehetnek ismétlődések, akkor egy δ műveletet is be kell iktatni az S-nek megfelelő kifejezés gyökerénél, hogy a kialakuló kifejezés ne állítson elő több sort, mint az eredeti lekérdezés.
  b) A kétargumentumú kiválasztást helyettesítsük egy σ_C egyargumentumú kiválasztással, ahol C az a feltétel, amelyet úgy kapunk, hogy a t sor minden egyes komponensét egyenlővé tesszük az S reláció neki megfelelő attribútumával.
  c) σ_C argumentumaként R és S szorzatát adjuk meg.

A következő ábra szemlélteti ezt a transzformációt:



Példa. Vegyük az előző példában előállított fát, és alkalmazzuk rá a fenti szabályt. Ezen az ábrán `R` a `SzerepelBenne` reláció, az `S` reláció pedig annak a relációalgebrai kifejezésnek az eredménye, amely a $\pi_{\text{név}}$ gyökerű részfából áll. A `t` sornak egy komponense van, nevezetesen a `színészNév` attribútum.

A kétargumentumú kiválasztás helyettesítője a $\sigma_{\text{színészNév} = \text{név}}$ kifejezés, amelynek `C` feltétele a `t` sor egyetlen tagját egyenlővé teszi az `S` lekérdezés eredményének egyetlen attribútumával. A σ csomópont gyereke egy $\times$ csomópont, amelynek gyerekei a `SzerepelBenne` címkéjű csomópont és az `S`-hez tartozó kifejezés gyökere. Mivel a `név` kulcsa a `FilmSzínész` relációnak, nincs szükség arra, hogy az `S`-hez tartozó kifejezésben bevezessünk egy ismétlődéseket megszüntető δ operátort. A következő ábra mutatja az új kifejezésfát, amely már teljesen relációalgebrában van kifejezve, és ekvivalens a rész első (alkérdést nem tartalmazó) példájában látható kifejezésfával, habár a szerkezete igencsak különböző:



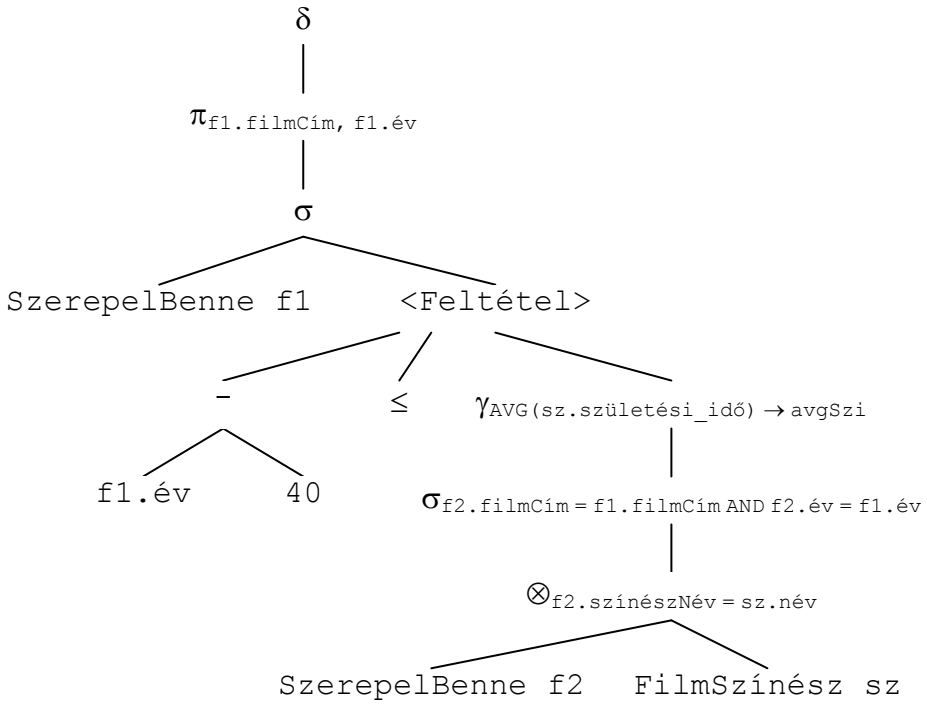
Összetettebb az alkérdések relációalgebrába történő átfordítása, ha az alkérdés korrelatív. Mivel a korrelatív alkérdések magukban foglalnak rajtuk kívül definiált, ismeretlen értékeket is, nem lehet őket külön átfordítani. Ehelyett az alkérdést úgy transzformáljuk, hogy az egy olyan relációt állít elő, amelyben bizonyos extra attribútumok is megjelennek: attribútumok, amelyeket később a kívül definiált attribútumokkal kell majd összehasonlítani. Az alkérdés attribútumait a külső attribútumokkal összevető feltételt erre a relációra alkalmazzuk, és az ezután már feleslegessé vált extra attribútumokat vetítés segítségével elhagyhatjuk. E folyamat során oda kell figyelnünk az ismétlődő sorok esetleges bevezetésére, amennyiben a lekérdezés a végén nem távolítja el az ismétlődéseket.

Példa. Vegyük az alábbi lekérdezést: „Adjuk meg azokat a filmeket és gyártási évüket, amelyekben szereplő színészek átlagéletkora legfeljebb 40 év volt, amikor a film készült!” A lekérdezés SQL-beli megfelelője a következő:

```
SELECT DISTINCT f1.filmCím, f1.év FROM SzerepelBenne f1
  WHERE f1.év - 40 <= (
    SELECT AVG(sz.születési_idő) FROM SzerepelBenne f2, FilmSzínész sz
      WHERE f2.színészNév = sz.név AND
        f2.filmCím = f1.filmCím AND f2.év = f1.év
  );
```

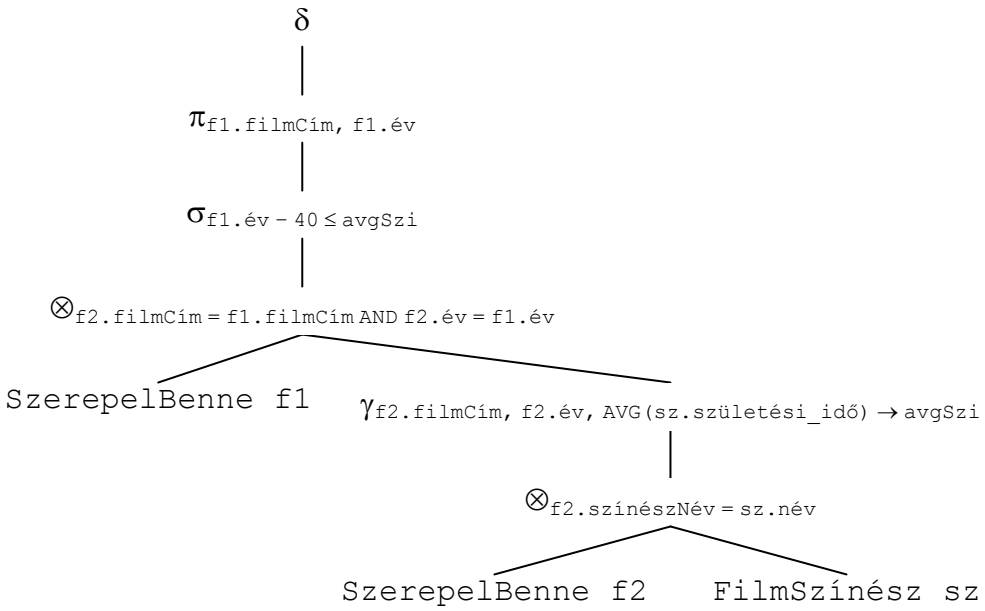
Az egyszerűség kedvéért a `születési_idő`-t születési évnek vesszük, így vehetjük azok átlagát, amit aztán a `SzerepelBenne` reláció `év` attribútumával össze tudunk hasonlítani. A lekérdezést úgy fogalmaztuk meg, hogy mindhárom hivatkozott reláció rendelkezik a maga saját sorváltozójával, mutatva, hogy a különböző attribútumok honnan származnak.

Az alábbi ábrán a lekérdezés elemzésének és a relációalgebrába való részleges átfordítás végrehajtásának az eredménye látható:



Ebben a kezdeti transzformációban kettéválasztottuk az alkérdés `WHERE` záradékát, és az egyik részt úgy használtuk, hogy relációk szorzatából összekapcsolást készítettünk. A sorváltozókat meghagytuk a fában is, hogy világos legyen az egyes attribútumok eredete. Megtehettük volna azt is, hogy vetítések segítségével átnevezzük az attribútumokat, de akkor az eredmény nehezebben lenne követhető.

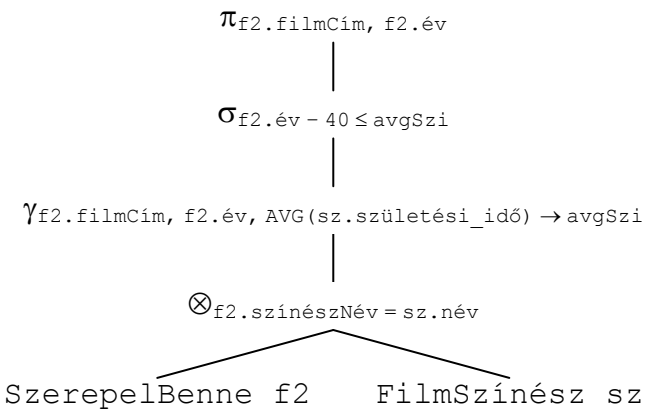
A `<Feltétel>` csomópont és a kétargumentumú kiválasztás eltávolításához szükség van egy olyan kifejezésre, amely a `<Feltétel>` jobb oldali ágához tartozó relációt definiálja. Az alkérdés azonban korrelatív, és az `f1.filmCím` és `f1.év` attribútumok nem szerezhetők meg az alkérdésben említett relációkból, amelyek az `f2` sorváltozójú `SzerepelBenne` és az `sz` sorváltozójú `FilmSzínész`. Emiatt a $\sigma_{\texttt{f2.filmCím = f1.filmCím AND f2.év = f1.év}}$ kiválasztást akkorra kell elhalasztani, amikor az alkérdés relációját már kombináltuk a `SzerepelBenne` relációnak a lekérdezés külső szintjén megjelenő példányával, azaz az `f1` sorváltozójú példánnyal. A logikai lekérdezésterv ilyen átalakításához módosítani kell a γ operátort úgy, hogy a csoportosítás az `f2.filmCím` és az `f2.év` attribútumok szerint történjen, így lesznek ugyanis elérhetők ezek az attribútumok a kiválasztáskor. Ennek hatásaként az alkérdéshez egy filmekből álló relációt számolunk ki, ahol minden egyes filmet annak címe és gyártási éve, valamint a filmben szereplő színészek születési évének átlaga képvisel. A módosított γ operátor az alábbi ábrán látható:



Az ábra a relációalgebrába történő teljes átfordítást is mutatja. A γ fölött a külső lekérdezésből származó `SzerepelBenne` relációnak és az alkérdés eredményének összekapcsolása szerepel. Az alkérdésben lévő kiválasztást a `SzerepelBenne` relációnak és az alkérdés relációjának szorzatára lehet alkalmazni, amit már egy théta-összekapcsolásként jelenítettünk meg, amivé ténylegesen válna az algebrai szabályok alkalmazása után. A théta-összekapcsolás fölött egy további kiválasztás szerepel, ami a külső lekérdezés kiválasztásának felel meg, ahol a filmek gyártási évét hasonlítjuk össze a színészek születési évének átlagával. Az algebrai kifejezés a fa tetején úgy végződik, mint az előző ábra kifejezése, vagyis a kívánt attribútumokra történő vetítéssel és az ismétlődések eltávolításával. A következő fejezetben látni fogjuk, hogy egy lekérdezésoptimalizáló sokkal többet is tehet a lekérdezésterv javítása érdekében. A jelenlegi konkrét példánkban teljesül három feltétel, amelyek lehetővé teszik, hogy a terven jelentősen javítsunk. Ezek a feltételek a következők:

1. Az ismétlődések megszüntetése a lekérdezés végén történik, nem pedig az összekapcsolás bal oldali argumentumában.
2. A vetítés a `SzerepelBenne f1` relációból kihagyja a színészek neveit, amelyet nem vezethetnénk be a `SzerepelBenne f2` relációból.
3. A `SzerepelBenne f1` és a maradék kifejezés közti összekapcsolás egyenlővé teszi a `SzerepelBenne f1` és a `SzerepelBenne f2` relációk `filmCím` és `év` attribútumait.

Mivel ezek a feltételek teljesülnek, az `f1.filmCím` és az `f1.év` összes előfordulását helyettesíthetjük `f2.filmCím`-mel, illetve `f2.év`-vel. Az ábrán a felső összekapcsolás ezáltal feleslegessé válik, csakúgy, mint a `SzerepelBenne f1` argumentum. Hasonlóan elhagyható a fa gyökerében az ismétlődések eltávolítása is, hiszen a csoportosítás nem állít elő ismétlődő sorokat, a vetítés pedig éppen a csoportosító attribútumokra vetít. Az így előálló logikai lekérdezésterv az alábbi ábrán látható:



Ha ezt a tervet visszaírjuk SQL-ben megfogalmazott kifejezéssé, a következőt kapjuk:

```
SELECT f2.filmCím, f2.év
  FROM SzerepelBenne f2, FilmSzínész sz
  WHERE f2.színészNév = sz.név
  GROUP BY f2.filmCím, f2.év
  HAVING f2.év – 40 <= AVG(sz.születési_idő);
```

Logikai lekérdezéstervek javítása

Amikor egy lekérdezést átfordítunk relációalgebrába, egy lehetséges logikai lekérdezéstervet kapunk, amit aztán a korábban felvázolt algebrai szabályok segítségével átírunk, végül a lekérdezésátíró egyetlen logikai lekérdezéstervet választ ki, amelyet a „legjobbnak vél” abban az értelemben, hogy a legolcsóbb fizikai lekérdezéstervet eredményezi.

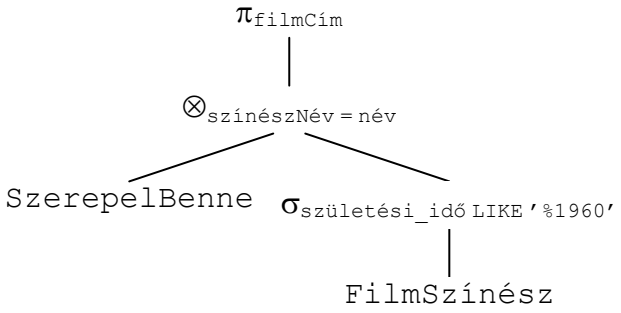
Nyitva hagyjuk viszont az „összekapcsolási sorrend” kérdését, így egy összekapcsolást tartalmazó logikai lekérdezésterv úgy tekinthető, mint azon tervek egy családja, amelyek megfelelnek azoknak a különböző lehetőségeknek, ahogyan egy összekapcsolás sorba rendezhető és csoportosítható. Az összekapcsolási sorrend megválasztását később tárgyaljuk. Ehhez hasonlóan, ha egy lekérdezésterv három vagy több relációt tartalmaz a többi kommutatív és asszociatív operátor argumentumaiként, akkor feltételezésünk szerint a logikai terv fizikai tervvé történő konvertálásakor átrendezés és átcsoportosítás megengedett. A sorrendet és a fizikai lekérdezésterv kiválasztását taglaló kérdésekkel a következő részben foglalkozunk.

Az előző részben számos olyan algebrai szabály szerepelt, amelyek feltehetően javítják a logikai tervet. Az optimalizálókban leggyakrabban használtak a következők:

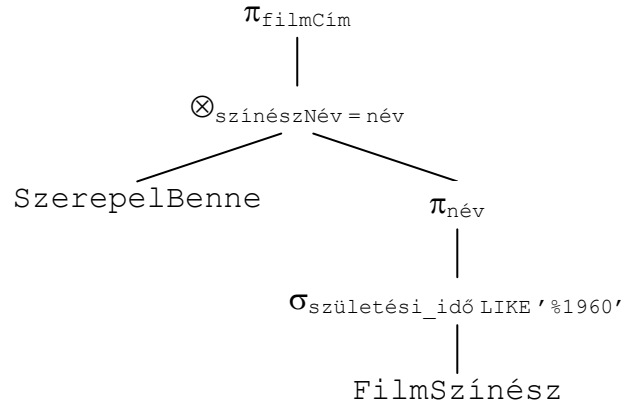
- A kiválasztásokat addig tologatjuk lefelé a fában, ameddig csak mehetnek. Ha egy kiválasztási feltétel több részfeltétel `AND` operátorral való összekapcsolása, akkor a feltételt szétvághatjuk, és az egyes részfeltételeket külön-külön vihetjük le a fában. Ez a stratégia valószínűleg a leghatékonyabb javítási technika, de nem árt szem előtt tartani a Kiválasztások tologatása című fejezetben mondottakat, miszerint bizonyos körülmények között a kiválasztást először a fa tetejére kell felvinni.
- Hasonlóképpen a vetítéseket is tologathatjuk lefelé a fában, vagyis új vetítéseket vezethetünk be. Csakúgy, mint a kiválasztás esetében, a vetítések tologatásával is óvatosan kell bánni, ahogy ezt a Vetítéssel kapcsolatos szabályok című fejezetben elmondtuk.
- Az ismétlődések megszüntetése néha eltávolítható, vagy áthelyezhető alkalmasabb helyre a fában az Ismétlődések elhagyására vonatkozó szabályok című fejezetben mondottaknak megfelelően.
- Bizonyos kiválasztások kombinálhatók egy alatta elhelyezkedő szorzattal úgy, hogy a műveletpár egy théta-összekapcsolássá alakul, amit általában sokkal hatékonyabban lehet kiértékelni, mint a két

műveletet külön-külön. Ezeket a szabályokat az Összekapcsolásra és szorzatra vonatkozó szabályok című fejezetben tárgyaltuk.

Példa. Vegyük a rész első példájában szereplő lekérdezést (28. oldal). Először vágjuk ketté a kiválasztást az AND operátor mentén. A második részfeltételt tartalmazó kiválasztás levihető a fában, mivel az egyetlen érintett attribútum (születési_idő) a FilmSzínész relációból származik. Az első részfeltétel a szorzat mindkét argumentumából tartalmaz egy-egy attribútumot, de azok egyenlővé vannak téve, ezért a szorzat és a kiválasztás együtt helyettesíthető egy egyenlőségen alapuló összekapcsolással. Az átalakítások eredményét az alábbi ábra mutatja:



Másik példaként tekintsük az előző fejezet első példájában elkészített kifejezésfát (30. oldal), amelyen szintén lehet javítani. Hasznos transzformációt azonban csak az előző példában is említett szabályok egyike jelent: egy kiválasztás és az alatta elhelyezkedő szorzat helyettesítése egy egyenlőségen alapuló összekapcsolással. A kapott lekérdezésterv a következő ábrán látható:



Ez a lekérdezésterv majdnem ugyanaz, mint az előző példában kapott terv, csak van benne egy további vetítés a név attribútumra vonatkozóan. Amikor az 1960-ban született színészek megkeresésére végrehajtunk egy kiválasztást a FilmSzínész reláción, elég, ha csak a név attribútumot állítjuk elő, mert ez az, amit a későbbi műveletekben használunk. Az utóbbi tervet az előbbiből is megkaphatjuk úgy, hogy a vetítést bevisszük a fa jobb oldali ágába (miközben a gyökérben is meghagyjuk). Ugyanakkor viszont a SzerepelBenne tárolt reláció vetítése költséges lehet, ha emiatt nem tudunk használni egy indexet a reláció azon sorainak elérésekor, amelyekre az összekapcsolásnál szükség van.

A kommutatív és asszociatív operátorok csoportosítása

A hagyományos elemzők nem állítanak elő olyan fákat, amelyek csomópontjai korlátlan számú gyerekkel rendelkezhetnek. Így az a normális, hogy az operátorok csak unáris vagy bináris formájukban jelennek meg. A kommutatív és asszociatív operátorok azonban felfoghatók úgy, mint amelyeknek tetszőleges számú operandusa van. Ha egy operátort többoperandusúként kezelünk, akkor lehetőséget kapunk az operandusok sorrendjének megváltoztatására. Ez azt eredményezheti, hogy az új sorrendnek megfelelő

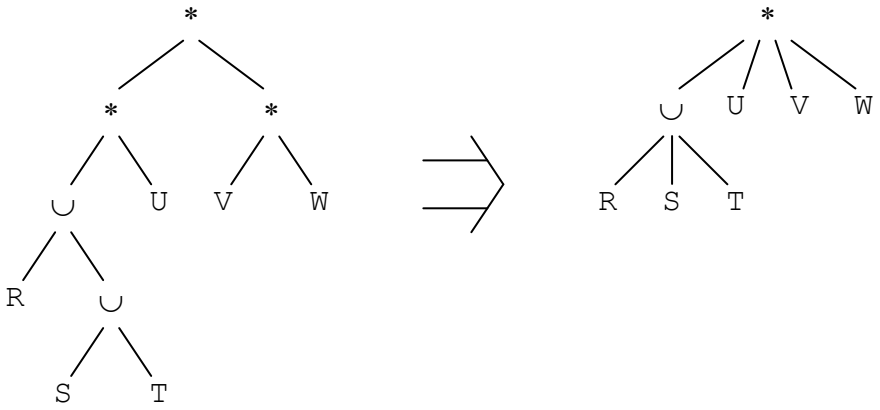
bináris műveletek sorozata kevesebb idő alatt hajtható végre, mint ha az elemzőfa által meghatározott sorrendben végeznénk el őket. A többoperandusú összekapcsolások rendezését később tárgyaljuk.

A végső logikai lekérdezésterv előállítása előtt tehát végrehajtunk egy utolsó lépést: ha van egy részfa, amelynek csomópontjaiban ugyanaz a kommutatív és asszociatív operátor szerepel, akkor ezeket a csomópontokat egyetlen, több gyerekkel rendelkező csomópontba csoportosítjuk. Emlékezzünk vissza, hogy a kommutatív és asszociatív operátorok a szorzat, a természetes összekapcsolás, az egyesítés és a metszet. Természetes összekapcsolások és théta-összekapcsolások is egyesíthetők egymással bizonyos körülmények között:

1. A természetes összekapcsolásokat olyan théta-összekapcsolásokkal kell helyettesíteni, amelyek egyenlővé teszik az azonos nevű attribútumokat.
2. Be kell iktatni egy vetítést az olyan attribútumok ismételt példányainak eltávolítására, amelyek a théta-összekapcsolássá vált természetes összekapcsolásban érintettek.
3. A théta-összekapcsolás feltételeinek asszociatívnak kell lenniük.

Ezenfelül a szorzatokat a természetes összekapcsolás speciális eseteként is felfoghatjuk, és egyesíthetjük őket összekapcsolásokkal, ha a fában egymás szomszédaiként helyezkednek el.

A következő ábra szemlélteti ezt a transzformációt egy olyan helyzetben, ahol a logikai lekérdezéstervben egy két egyesítésből álló nyaláb, valamint egy három összekapcsolásból álló nyaláb szerepel (a betűk `R`-től `W`-ig kifejezéseket jelölnek, nem feltétlenül tárolt relációkat):



Műveletek költségének becslése

Tegyük fel, hogy elemeztünk egy lekérdezést, és átalakítottuk egy logikai lekérdezéstervvé. Tegyük fel továbbá, hogy elvégeztük a kiválasztott transzformációkat, és megkaptuk a legjobbnak vélt logikai lekérdezéstervet. Következő lépésként a logikai tervet kell fizikai tervvé alakítani. Ez általában úgy történik, hogy sok különböző fizikai tervet tekintünk, amelyek a logikai tervből származnak, és kiértékeljük vagy becsüljük az ezekhez tartozó költségeket. E kiértékelés után, amit *költség alapú felsorolásnak* nevezünk, kiemeljük a legkisebb költségű fizikai tervet, és azt adjuk tovább a lekérdezés-végrehajtó motornak. Amikor az egy adott logikai lekérdezéstervből levezethető lehetséges fizikai terveket felsoroljuk, az egyes fizikai tervekhez az alábbiakat is kiválasztjuk:

1. Sorrendiség és csoportosítás a kommutatív-asszociatív operátorokra (szorzat, összekapcsolás, egyesítés, metszet) vonatkozóan.
2. Algoritmus a logikai tervben szereplő minden egyes operátorhoz. (Például beágyazott ciklusú vagy tördelő összekapcsolást használjunk-e.)
3. További műveletek (például beolvasás, rendezés), amelyek a fizikai tervhez szükségesek, de a logikai tervben explicit módon nem szerepelnek.
4. Annak módja, ahogy egy operátor továbbadja az argumentumokat egy másiknak. (Például lemezen történő tárolással, iterátorokat használva vagy központi memóriapufferben.)

A továbbiakban megvizsgáljuk mindezeket a kérdéseket. Annak érdekében azonban, hogy az ezekkel a választásokkal kapcsolatban felmerülő kérdéseket megválaszolhassuk, meg kell értenünk, hogy a különböző fizikai tervek költsége mit is jelent. Egy terv pontos költségét nem tudhatjuk meg a terv végrehajtása nélkül, és egy lekérdezéshez nyilván nem akarunk egynél több tervet végrehajtani. Így a tervek költségének becslésére kényszerülünk, anélkül hogy azokat végrehajtanánk.

Mielőtt elkezdjük a fizikai tervek felsorolásának tárgyalását, az ilyen tervek költségbecslésének mikéntjére kell kitérni. Ezek a becslések az adatokkal kapcsolatos paraméterekre épülnek, amelyeket vagy pontosan kiszámítunk az adatokból, vagy a – hamarosan ismertetésre kerülő – „statisztikai gyűjtés" nevű eljárással becslünk. Ha adottak a paramétereknek az értékei, akkor számos elfogadható becslés adható a relációméretekkel kapcsolatban, amelyekből aztán a teljes fizikai terv költsége megbecsülhető.

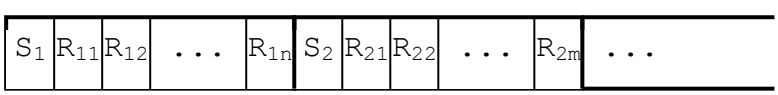
A relációk méretét jellemző paraméterek a következők:

- `B(R)` jelöli az `R` reláció összes sorának tárolásához szükséges blokkok számát.
- `T(R)` jelöli az `R` reláció sorainak számát, azaz `R` *számosságát*.
- `V(R,a)` jelöli az `R` reláció `a` attribútumához tartozó *értékszámlálót*, vagyis azoknak a különböző értékeknek a számát, amelyek az `R` relációban az `a` attribútum értékeként előfordulnak. Hasonlóan: $V(R,[a_1, a_2, \ldots, a_n])$ jelöli azoknak a különböző értékkombinációknak a számát, amelyek előfordulnak `R`-ben, amikor az $a_1, a_2, \ldots, a_n$ attribútumokat együtt tekintjük, vagy másképpen a $\pi_{a1, a2, \ldots, an}(R)$-ben szereplő különböző sorok számát jelenti, azaz $T(\delta(\pi_{a1, a2, , \ldots, an}(R)))$.

Közbülső relációk méretének becslése

A fizikai tervet úgy választjuk ki, hogy a lekérdezés végrehajtásának költsége minimális legyen. A legfőbb költségtényező rendszerint a lemez I/O-műveletek száma, de néha fontos a processzoridő és – ha a lekérdezést párhuzamosan, például több egymással összekötött gépen hajtjuk végre – a kommunikációs idő is.

Egy reláció sorainak beolvasásához szükséges lemez I/O-műveletek száma erősen függ attól, hogy milyen módon tároljuk a relációt. A következőkben leírtak megértéséhez be kell vezetnünk a nyalábolás fogalmát. Egy relációt akkor nevezünk *nyaláboltnak* (clustered), ha a reláció sorait olyan blokkokban tároljuk, amelyek kizárólag vagy főként a szóban forgó reláció tárolására hivatottak. Nem keverendő a fogalom a nyalábolt fájlszervezéssel, ami azt jelenti, hogy egy `R` reláció bizonyos sorait együtt tároljuk egy másik `S` reláció azon sorával, amely a közös attribútumokon megegyezik `R` ezen soraival. Például a filmeket tároló reláció (`R`) azonos stúdióhoz tartozó sorait csoportosíthatjuk a stúdiókat tároló reláció (`S`) azon sorával, amely az adott filmeket készítő stúdió adatait tartalmazza. Ebben az esetben a blokkok „főként" `R` számára vannak fenntartva, tehát `R` nyalábolt reláció, míg a stúdiókat tartalmazó `S` reláció nem nyalábolt, hiszen sorai főként inkább `R`-beli soroknak szánt blokkokban találhatók:



Amikor a logikai terv kifejezése több operátort tartalmaz, bizonyos dolgokat tudunk arról, hogy a közbülső relációk hogyan lesznek ábrázolva. Amíg ugyanis a kifejezés argumentumaiként szolgáló tárolt relációk többféleképpen lehetnek tárolva (nyalábolatlan vagy nem, indexelve vagy index nélkül), a lekérdezés végrehajtása közben kiszámított valamely reláció, amelyet lemezen tárolunk, tárolható nyalábolatlan úgy, hogy minél kevesebb blokkot foglaljon el. Másrészt egy ilyen relációnak nem lesznek indexei, hacsak nem definiáljuk azokat explicit módon a fizikai lekérdezésterv részeként.

Ezek után azt mondhatjuk, hogy a köztes relációk kezeléséhez szükséges lemez I/O-műveletek száma nem függ mástól, mint a relációk méretétől. Ezt pedig úgy kapjuk meg, hogy a közbülső reláció sorainak

számát megszorozzuk az egy sor tárolásához szükséges bájtok számával. Az egy sor által elfoglalt bájtok száma levezethető a közbülső reláció attribútumaiból és azok típusából, így csak az marad rejtély, hogy a köztes reláció hány sort tartalmaz. Mivel általában nem tudjuk pontosan megmondani, hogy egy köztes relációnak hány sora lesz, néhány ésszerű szabályt fogunk bevezetni ezeknek a méreteknek a becslésére.

Ideális esetben egy közbülső relációban szereplő sorok számát becslő szabályokra igazak az alábbiak:

1. Elég pontos becslést adnak.
2. Könnyű kiszámolni.
3. Logikailag konzisztensek, azaz egy közbülső reláció méretére vonatkozó becslés nem függ a reláció kiszámításának módjától. Például egy több reláció összekapcsolására vonatkozó becslés nem függ a relációk összekapcsolásának sorrendjétől.

Nincs általános egyetértés e feltételek teljesítésére vonatkozóan. Mi bemutatunk néhány egyszerű szabályt, amelyek a legtöbb helyzetben megfelelők. Szerencsére a méret becslésének nem az a célja, hogy a pontos méretet előre kiszámítsuk, hanem az, hogy hozzájáruljon egy fizikai terv kiválasztásához. Még egy pontatlan méretbecslési módszer is szolgálhat erre a célra, ha konzisztens módon hibázik, azaz ha a legjobb fizikai tervhez rendeli a legkisebb költséget, még ha annak a tervnek a tényleges költségéről az derül is ki, hogy különbözik az előre kiszámítottól.

A vetítések méretének becslése

A vetítés abban különbözik a többi művelettől, hogy az eredményének a mérete kiszámítható. Mivel egy vetítés az argumentuma minden sorához előállít egy eredménysort, a kimenet mérete a bemenet méretéhez képest csak a sorok hosszában különbözik. Az általunk használt vetítés operátor ugyanis multihalmaz operátor, és nem távolítja el az ismétlődéseket. Ha a vetítés során előálló ismétlődéseket meg akarjuk szüntetni, akkor a δ operátort kell utána alkalmazni.

Normális esetben vetítéskor a sorok összezsugorodnak, hiszen bizonyos komponenseket elhagyunk. A vetítés általános formája azonban megengedi új komponensek létrehozását is. Vannak tehát esetek, amikor egy π operátor növeli egy reláció méretét.

Példa. Tegyük fel, hogy `R(a,b,c)` egy reláció, ahol `a` és `b` négybájtos egészek, `c` pedig 100 bájtos karakterlánc. Tegyük fel továbbá, hogy a sorfejlécek 12 bájtot igényelnek. Ekkor az `R` reláció minden egyes sorának 120 bájtra van szüksége. Legyenek a blokkok 1024 bájt hosszúak, 24 bájtos blokkfejlécekkel. Egyetlen blokkban így 8 sor fér el. Tegyük fel, hogy `T(R)` = 10000, vagyis hogy `R` 10000 sort tartalmaz. Ekkor `B(R)` = 1250.

Legyen $S = \pi_{a+b \to x,\, c}(R)$, azaz `a`-t és `b`-t az összegükkel helyettesítjük. `S` sorai 116 bájtot igényelnek: 12-t a fejlécnek, 4-et az összegnek és 100-at a karakterláncnak. Bár `S` sorai valamivel kisebbek, mint `R` sorai, még mindig csak 8 sort helyezhetünk egy blokkba, tehát `T(S)` = 10000 és `B(S)` = 1250.

Legyen $U = \pi_{a,\, b}(R)$, azaz elhagyjuk a karakterlánc komponenst. `U` sorai csak 20 bájt hosszúak. `T(U)` még mindig 10000. Most azonban `U`-nak 50 sorát pakolhatjuk egy blokkba, vagyis `B(U)` = 200. Ez a vetítés tehát a relációt mintegy hatodrészére zsugorítja.

A kiválasztások méretének becslése

Amikor egy kiválasztást hajtunk végre, általában csökkentjük a sorok számát, de a sorok mérete ugyanaz marad. A kiválasztás legegyszerűbb esetében, amikor egy attribútumnak egy konstanssal való egyenlőségét vizsgáljuk, létezik egy könnyű módszer az eredmény méretének becslésére, feltéve, hogy

tudjuk (vagy becsülni tudjuk) az attribútum által felvett különböző értékek számát. Legyen $S = \sigma_{a=c}(R)$, ahol a az R reláció egy attribútuma, c pedig egy konstans. Ekkor a következő becslés használható:

- $T(S) = T(R) / V(R,a)$

Ez a szabály biztosan igaz akkor, ha az a attribútum minden értéke egyenlő gyakorisággal fordul elő a relációban. Belátható, hogy még akkor is ez a szabály adja a legjobb becslést, ha a értékei nem mutatnak egyenletes eloszlást a relációban. Elvárjuk viszont, hogy a minden értéke egyforma valószínűséggel szerepeljen az a értékét meghatározó lekérdezésekben.

Problematikusabb a méret becslése, ha a kiválasztási feltétel egyenlőtlenségen alapuló összehasonlítást tartalmaz, például ha $S = \sigma_{a<10}(R)$. Azt gondolhatnánk, hogy az átlag tekintetében a sorok fele megfelelne az összehasonlításnak, így $T(R)/2$ jó becslése lenne S méretének. Intuíció alapján azonban egy ilyen lekérdezés a lehetséges soroknak inkább csak egy kisebb hányadát adná vissza. Ezért egy olyan szabályt vezetünk be, amely figyelembe veszi ezt a tendenciát, és azzal a feltételezéssel él, hogy egy tipikus egyenlőtlenségen alapuló vizsgálat körülbelül a sorok egyharmadát adja vissza, nem a felét. Így ha $S = \sigma_{a<c}(R)$, akkor $T(S)$-re a becslésünk:

- $T(S) = T(R) / 3$

A „nem egyenlő” összehasonlítások ritkák. Ha egy olyan kiválasztással találkozunk, mint például az $S = \sigma_{a \neq 10}(R)$, akkor feltételezhetjük, hogy lényegében minden sor kielégíti ezt a feltételt. Vehetjük tehát becslésként a következőt:

- $T(S) = T(R)$

Egy másik becslés lehet az alábbi:

- $T(S) = T(R) \cdot (V(R,a) - 1) / V(R,a)$

Ez a megközelítés valamivel kevesebbet ad, mint az előző becslés, mivel elismeri, hogy az R reláció sorainak körülbelül $1/V(R,a)$ része elbukik a feltételen, mert azokban a értéke egyenlő a konstanssal.

Amikor egy C kiválasztási feltétel több AND-del összekötött egyenlőségvizsgálat vagy más összehasonlítás, akkor a $\sigma_C(R)$ kiválasztást úgy tekinthetjük, mint azoknak az egyszerű kiválasztásoknak egymás utáni alkalmazását, amelyek mindegyike a feltétel egy-egy részét ellenőrzi. Ezeknek a kiválasztásoknak a sorrendje nem számít. Ennek hatásaként az eredmény méretére vonatkozó becslést úgy kapjuk meg, hogy az eredeti reláció méretét megszorozzuk az egyes feltételekhez tartozó *szelektivitási tényezőkkel*. Ez a tényező 1/3 egyenlőtlenség esetén, 1 „nem egyenlő” esetén, illetve $1/V(R,a)$, amikor a C feltételben egy a attribútum egyenlőségét vizsgáljuk egy konstanssal.

Példa. Legyen $R(a,b,c)$ egy relácó és $S = \sigma_{a=10 \text{ AND } b<20}(R)$. Legyen továbbá $T(R) = 10000$ és $V(R,a) = 50$. Ekkor a legjobb becslés $T(S)$-re: $T(R) / (50 \cdot 3) = 67$. Vagyis R sorainak 1/50 része éli túl az $a = 10$ szűrőt, és 1/3 része éli túl a $b < 20$ szűrőt.

Egy érdekes speciális eset, ami romba dönti az analízisünket, amikor a feltétel ellentmondásos. Nézzük például az $S = \sigma_{a=10 \text{ AND } a>20}(R)$ kiválasztást. Ekkor a szabályunk alapján $T(S) = 67$, ugyanakkor viszont világos, hogy egyetlen sorra sem teljesülhet a két részfeltétel mindegyike, tehát a helyes válasz: $T(S) = 0$. A logikai lekérdezésterv átírásakor a lekérdezésoptimalizáló sok speciális esetre vonatkozó szabályt figyelembe tud venni. A fenti esetben alkalmazhat egy olyan szabályt, amely a kiválasztási feltételt azonosan hamisnak találja, és az S-nek megfelelő kifejezést az üres halmazzal helyettesíti.

Amikor egy kiválasztás OR-ral kapcsolt részfeltételeket tartalmaz, kevesebb bizonyosságunk van az eredmény méretét illetően. Egy $S = \sigma_{C1 \text{ OR } C2}(R)$ alakú kiválasztás esetén egyszerű feltételezés az, hogy egyetlen sorra sem teljesül mindkét feltétel, vagyis az eredmény mérete egyenlő az egyes feltételeket

kielégítő sorok számának összegével. Ez a becslés általában túlbecslést jelent, és néha ahhoz az abszurd következtetéshez vezethet, hogy S-ben több sor van, mint R-ben. Ezért egy másik egyszerű megközelítés lehet, hogy vesszük a minimumát R méretének, és a C_1-t és C_2-t kielégítő sorok számának az összegének.

Egy kevésbé egyszerű, de feltehetően pontosabb becslést kapunk S méretére, ha feltesszük, hogy C_1 és C_2 függetlenek. Ekkor ha R-nek n sora van, amelyek közül m_1-re teljesül C_1, és m_2-re teljesül C_2, akkor az S-ben megjelenő sorok számára a következő becslést adhatjuk:

- $T(S) = n \cdot (1 - (1 - m_1 / n) \cdot (1 - m_2 / n))$

Itt az $1 - m_1 / n$ egyenlő a sorok C_1-et nem teljesítő hányadával, $1 - m_2 / n$ pedig a sorok C_2-t nem teljesítő hányadát jelenti. Ezek szorzata R azon sorainak hányadát adja, amelyek nincsenek benne S-ben, így ezt a szorzatot 1-ből kivonva az S-ben szereplő sorok hányadát kapjuk.

Példa. Tegyük fel, hogy az R(a,b) relációnak $T(R) = 10000$ sora van, és legyen $S = \sigma_{a = 10 \text{ OR } b < 20}(R)$. Legyen $V(R,a) = 50$. Ekkor az $a = 10$ feltételt kielégítő sorok számát $T(R) / V(R,a) = 200$-ra, a $b < 20$ feltételt kielégítő sorok számát $T(R) / 3 = 3333$-ra becsüljük.

Az S méretére vonatkozó legegyszerűbb becslés ezek összege, azaz 3533. A két részfeltétel függetlenségére építő bonyolultabb becslés a $10000 \cdot (1 – (1 – 200 / 10000) \cdot (1 – 3333 / 10000)) = 3466$ értéket adja. A két becslés között kicsi az eltérés, így nagyon valószínűtlen, hogy az egyik választása a másikkal szemben változást jelentene a legjobb fizikai terv kiválasztásában.

Az utolsó operátor, amely egy kiválasztási feltételben szerepelhet, a NOT. Ha egy R relációnak T(R) számú sora van, akkor a NOT C feltételt kielégítő sorok becsült számát úgy kapjuk meg, hogy T(R)-ből kivonjuk a C-t kielégítő sorok becsült számát.

Az összekapcsolások méretének becslése

Csak a természetes összekapcsolást fogjuk vizsgálni. A többi összekapcsolás az alábbi elveknek megfelelően kezelhető:

1. Egy egyenlőségen alapuló összekapcsolás eredményében megjelenő sorok száma pontosan úgy számítható ki, mint természetes összekapcsolás esetén (miután elszámoltunk a változónevekben bekövetkező változásokkal). (Lásd egy későbbi példában.)
2. Más théta-összekapcsolások úgy becsülhetők, mintha szorzatot követő kiválasztások volnának, figyelembe véve a következő megjegyzéseket:
   a) Egy szorzat sorainak számát úgy kapjuk meg, hogy összeszorozzuk a szorzatban részt vevő relációk sorainak számát.
   b) Egy egyenlőséget vizsgáló összehasonlítást a természetes összekapcsoláshoz kidolgozott technika segítségével becsülhetünk.
   c) Egy két attribútum egyenlőtlenségét vizsgáló, R.a < S.b alakú összehasonlítás úgy kezelhető, mint egy R.a < 10 alakú összehasonlítás, vagyis feltehetjük, hogy ennek a feltételnek a szelektivitási tényezője 1/3 (ha úgy gondoljuk, hogy a feltétel ritkán teljesül), vagy lehet 1/2 (ha nem élünk ezzel a feltételezéssel).

Első körben tételezzük fel, hogy két reláció természetes összekapcsolása csak két attribútum egyenlőségét tartalmazza. Ez azt jelenti, hogy az R(X,Y) * S(Y,Z) összekapcsolást vizsgáljuk, ahol X, Y és Z attribútumhalmazok, de kezdetben feltesszük, hogy Y egyetlen attribútumból áll: Y = {y}.

Az a probléma, hogy nem tudjuk, hogy R-ben és S-ben az y attribútum értékei milyen viszonyban állnak egymással. Például:

1. A két relációban az y attribútum értékei alkothatnak diszjunkt halmazokat. Ekkor az összekapcsolás eredménye üres reláció, és T(R * S) = 0.
2. Az y attribútum lehet elsődleges kulcs S-ben és idegen kulcs R-ben. Ilyenkor R minden egyes sora pontosan egy S-beli sorral kapcsolódik, ha y R-ben nem vesz fel NULL értéket, így T(R * S) = T(R).
3. Előfordulhat, hogy R és S majdnem minden sorában ugyanaz y értéke. Ekkor T(R * S) körülbelül T(R) · T(S) lesz.

A következő két egyszerűsítő feltételezéssel fogunk élni, hogy a leggyakoribb eseteket tekinthessük:

1. *Értékhalmazok tartalmazása*: Ha y egy több relációban is szereplő attribútum, akkor y összes előfordulása mindegyik relációban egy y_1, y_2, … rögzített értéklistának az elejéről kap értéket. Következésképpen: ha R és S két reláció, amelyek tartalmazzák az y attribútumot, és V(R,y) ≤ V(S,y), akkor y-nak az összes R-beli értéke S-ben is megjelenik. Nyilván előfordulhat, hogy ez a feltevés nem teljesül, de biztosan igaz például akkor, ha y az S relációban elsődleges kulcs, R-ben pedig idegen kulcs. Sok más esetben is megközelítőleg igaz, hiszen intuitíve azt várjuk, hogy ha S-ben y-nak sok különböző értéke van, akkor y egy adott R-beli értéke jó eséllyel szerepel S-ben.
2. *Értékhalmazok megőrzése*: Ha egy R relációt összekapcsolunk egy másik relációval, akkor egy a attribútum, amely nem összekapcsolási attribútum (azaz nem szerepel mindkét relációban), nem „veszít el” értékeket a lehetséges értékeinek halmazából. Pontosabban: ha a R-nek attribútuma, de S-nek nem, akkor V(R * S,a) = V(R,a). (Természetesen R és S összekapcsolásának sorrendje nem lényeges, tehát V(S * R,a) = V(R,a) is igaz.) Ez a feltételezés is sérülhet, de biztosan igaz akkor, ha R * S összekapcsolási attribútuma kulcs az S relációban, és idegen kulcs R-ben, ahol nem vesz fel NULL értéket. Csak akkor fordulhat elő, hogy ez az előfeltevés nem teljesül, ha R-nek lógó sorai vannak (azaz olyan sorok, amelyek S egyetlen sorával sem kapcsolódnak), de még ebben az esetben is érvényes lehet a feltétel.

E feltételezések mellett az R(X,y) * S(y,Z) összekapcsolás mérete a következőképpen becsülhető: Legyen V(R,y) ≤ V(S,y). Ekkor 1/V(S,y) az esélye annak, hogy R egy t sora S egy adott sorával kapcsolódik. Mivel S-nek T(S) sora van, azoknak a soroknak a várható száma, amelyekkel t kapcsolódik, T(S) / V(S,y). Minthogy R-nek T(R) sora van, R * S becsült mérete: T(R) · T(S) / V(S,y). Ha V(S,y) ≤ V(R,y), akkor a szimmetria alapján kapott becslés: T(R * S) = T(R) · T(S) / V(R,y). Általánosságban V(R,y) és V(S,y) közül a nagyobbal osztunk, tehát a végleges szabály a következő:

- T(R * S) = T(R) · T(S) / max(V(R,y), V(S,y))

Példa. Tekintsük a következő három relációt és lényeges paramétereiket:

R(a,b)	S(b,c)	U(c,d)
T(R) = 1000	T(S) = 2000	T(U) = 5000
V(R,b) = 20	V(S,b) = 50	
	V(S,c) = 100	V(U,c) = 500

Tegyük fel, hogy az R * S * U természetes összekapcsolást szeretnénk kiszámítani. Ennek egy lehetséges csoportosítási módja: (R * S) * U. Először tehát T(R * S)-t adjuk meg, ami 1000 · 2000 / 50 = 40000.

Ezután jön az R * S összekapcsolása U-val. Az eredmény méretére vonatkozó becslésünk: T(R * S) · T(U) / max(V(R * S,c), V(U,c)). A második előfeltevésünk alapján V(R * S,c) = V(S,c) = 100, vagyis az összekapcsolás során a c attribútum egyetlen értéke sem tűnik el. Ezek alapján a végső becslésünk: T((R * S) * U) = 40000 · 5000 / 500 = 400000.

Kezdhetnénk S és U összekapcsolásával is. Ekkor azt a becslést kapjuk, hogy $T(S * U) = 2000 \cdot 5000 / 500 = 20000$. Az előfeltevésünk alapján $V(S * U, b) = V(S, b) = 50$, így az eredmény méretének becslése: $T(R * (S * U)) = 1000 \cdot 20000 / 50 = 400000$.

Nem véletlen, hogy az $R * S * U$ méretére ugyanazt a becslést kapjuk, függetlenül attól, hogy melyik összekapcsolással kezdjük. Korábban megfogalmaztunk egy kívánalmat, mégpedig azt, hogy egy kifejezés eredményére vonatkozó becslés ne függjön a kiértékelés sorrendjétől. Belátható, hogy a két előfeltevésünk garantálja, hogy egy természetes összekapcsolásra vonatkozó becslés ugyanaz lesz, függetlenül az összekapcsolások végrehajtási sorrendjétől.

Természetes összekapcsolás több összekapcsolási attribútummal

Most nézzük meg, mi történik akkor, amikor az $R(X, Y) * S(Y, Z)$ összekapcsolásban az Y több attribútumot jelöl. Tegyük fel először, hogy Y két attribútumból áll: $Y = \{y_1, y_2\}$, azaz az $R(X, y_1, y_2) * S(y_1, y_2, Z)$ összekapcsolást szeretnénk végrehajtani. Annak valószínűsége, hogy R egy r sora S egy s sorával kapcsolódik, a következőképpen számítható ki:

Először megnézzük, mi a valószínűsége annak, hogy r és s megegyeznek az y_1 attribútumon. Tegyük fel, hogy $V(R, y_1) \geq V(S, y_1)$. Ekkor – az értékhalmazok tartalmazásáról szóló előfeltevés alapján – az s sorban y_1 értéke biztosan y_1 R-ben előforduló értékeinek valamelyike. Így annak esélye, hogy r-ben ugyanaz y_1 értéke, mint s-ben, $1/V(R, y_1)$. Ha $V(R, y_1) \leq V(S, y_1)$, akkor pedig y_1 r-beli értéke szerepelni fog S-ben, és $1/V(S, y_1)$ a valószínűsége annak, hogy y_1 értéke ugyanaz lesz r-ben és s-ben. Általánosságban azt mondhatjuk, hogy $1/\max(V(R, y_1), V(S, y_1))$ annak valószínűsége, hogy a két sorban y_1 értékei megegyeznek.

Hasonló gondolatmenet alapján állíthatjuk, hogy $1/\max(V(R, y_2), V(S, y_2))$ annak a valószínűsége, hogy r-ben és s-ben y_2 értékei megegyeznek. Mivel y_1 és y_2 értékei függetlenek, annak a valószínűsége, hogy a két sor mind a két attribútumon megegyezik, e két tört szorzata lesz. Ezek alapján az R és S soraiból képzett $T(R) \cdot T(S)$ darab sorpár közül az y_1 és y_2 attribútumokon egyező sorpárok száma:

- $T(R * S) = T(R) \cdot T(S) / (\max(V(R, y_1), V(S, y_1)) \cdot \max(V(R, y_2), V(S, y_2)))$

Ha a két relációnak tetszőleges számú közös attribútuma van, akkor a következő szabály alkalmazható egy természetes összekapcsolás méretének becslésére:

- $T(R * S) = T(R) \cdot T(S) / (\max(V(R, y_1), V(S, y_1)) \cdot \ldots \cdot \max(V(R, y_n), V(S, y_n)))$,

ahol $y_1, \ldots, y_n$ a közös attribútumok.

Példa. A következő példa a fenti szabályt alkalmazza, egyben azt is szemlélteti, hogy a természetes összekapcsolásra vonatkozó eddigi eredményeink az egyenlőség alapú összekapcsolásokra is érvényesek. Tekintsük az alábbi összekapcsolást:

$R(a,b,c) \otimes_{R.b = S.d \text{ AND } R.c = S.e} S(d,e,f)$

A méretekkel kapcsolatos paraméterek legyenek a következők:

R(a,b,c)	S(d,e,f)
T(R) = 1000	T(S) = 2000
V(R,b) = 20	V(S,d) = 50
V(R,c) = 100	V(S,e) = 50

Ez az összekapcsolás természetes összekapcsolásként is felfogható, ha az R.b és S.d attribútumokat, illetve az R.c és S.e attribútumokat közös attribútumoknak tekintjük. Ekkor a fenti szabály alapján az R * S méretének becsült értéke: T(R * S) = 1000 · 2000 / (50 · 100) = 400 sor.

Másik példaként tekintsük újra az előző fejezet végén vizsgált összekapcsolást. Vegyük most a harmadik lehetséges összekapcsolási sorrendet, amikor is először az R(a,b) * U(c,d) összekapcsolást számítjuk ki. Ez az összekapcsolás valójában egy szorzat, így az eredményben előálló sorok száma: T(R) · T(U) = 1000 · 5000 = 5000000. A szorzatban a különböző b-k száma V(R * U,b) = V(R,b) = 20, a különböző c-k száma pedig V(R * U,c) = V(U,c) = 500.

Amikor ezt a szorzatot S(b,c)-vel összekapcsoljuk, a méret becsléséhez összeszorozzuk a két relációban a sorok számát, majd osztunk max(V(R * U,b), V(S,b))-vel és max(V(R * U,c), V(S,c))-vel. Az így kapott mennyiség: T(S * (R * U)) = 2000 · 5000000 / (50 · 500) = 400000. Láthatjuk, hogy az összekapcsolásnak ez a harmadik sorrendje az eredmény méretére ugyanazt a becslést adja, mint a másik két esetben.

Habár a relációk méretére vonatkozó vizsgálódásaink során az eredményben szereplő sorok számára összpontosítottunk, az egyes sorok méretét is számításba kell venni. Relációk összekapcsolása például az eredeti relációkban előforduló soroknál nagyobb sorokat állít elő. Egy R * S összekapcsolás, ahol mindkét reláció 1000 sort tartalmaz, adhat olyan eredményt, amely szintén 1000 sorból áll. Az eredmény azonban több blokkot foglalna el, mint R vagy S.

A fenti példa egy érdekes eset ezzel kapcsolatban. Egy théta-összekapcsoláskor kapott sorok számának becslésére használhatunk ugyan természetes összekapcsolásra vonatkozó technikákat, ahogy ott tettük is, de egy théta-összekapcsolás több komponensből álló sorokat állít elő, mint a neki megfelelő természetes összekapcsolás. A konkrét példában szereplő théta-összekapcsolás hat komponensből álló sorokat eredményez, míg az R(a,b,c) * S(b,c,d) természetes összekapcsolás ugyanannyi sort állít ugyan elő, de a soroknak csak négy komponense lesz.

Sok reláció összekapcsolása

Vizsgáljuk meg végül a természetes összekapcsolás általános esetét:

$S = R_1 * R_2 * \ldots * R_n$

Tegyük fel, hogy az a attribútum az R_i-k közül k-ban fordul elő, és hogy ebben a k relációban a különböző értékeire az alábbi egyenlőtlenség teljesül: $v_1 \le v_2 \le \ldots \le v_k$, ahol $v_i = V(R_i,a)$. Vegyünk mindegyik relációból egy sort. A kérdés, hogy mennyi a valószínűsége annak, hogy a kiválasztott sorok mindegyike megegyezik az a attribútumon.

Tekintsük azt a t_1 sort, amelyiket abból az R_1 relációból választottunk, amelyben a különböző értékeinek száma a legkisebb, azaz v_1. Az értékhalmazok tartalmazására vonatkozó előfeltevés alapján a v_1 számú érték mindegyike az a attribútummal rendelkező összes többi relációban megtalálható. Vegyük az R_i relációt. Az ebből a relációból vett t_i sor $1/v_i$ valószínűséggel egyezik meg t_1-gyel az a attribútumon. Mivel ez a kijelentés minden $i = 2, 3, \ldots, k$ esetén igaz, annak valószínűsége, hogy mind a k sor megegyezik az a attribútumon, ezen hányadosok szorzata lesz, vagyis $1/v_2v_3\ldots v_k$. Ez az eredmény vezet el a tetszőleges összekapcsolás méretének becslésére vonatkozó szabályhoz:

- Vegyük először az egyes relációkban szereplő sorok számának szorzatát, majd osszuk ezt el a legkisebb kivételével az összes V(R,a)-val minden olyan a attribútumra, amely legalább kétszer előfordul a relációkban.

Az összekapcsolás után az a attribútum értékeként megmaradó értékek számát szintén becsülhetjük. Az értékhalmazok megőrzésére vonatkozó előfeltevés alapján ez a fenti V(R,a)-k legkisebbike lesz.

Példa. Vegyük az R(a,b,c) * S(b,c,d) * U(b,e) összekapcsolást a következő lényeges paraméterekkel:

R(a,b,c)	S(b,c,d)	U(b,e)
T(R) = 1000	T(S) = 2000	T(U) = 5000
V(R,a) = 100		
V(R,b) = 20	V(S,b) = 50	V(U,b) = 200
V(R,c) = 200	V(S,c) = 100	
	V(S,d) = 400	
		V(U,e) = 500

Az eredmény méretének becsléséhez először képezzük a relációk méreteinek szorzatát: $1000 \cdot 2000 \cdot 5000$. Ezután megnézzük, hogy mely attribútumok szerepelnek egynél többször: ezek a b, amely háromszor fordul elő, és a c, amely kétszer. Ezek alapján a szorzatot elosztjuk a V(R,b), a V(S,b) és a V(U,b) közül a két legnagyobbal, ami 50 és 200. Végül tovább osztunk a V(R,c) és a V(S,c) közül a nagyobbal, ami 200. A kapott becslés tehát: $1000 \cdot 2000 \cdot 5000 / (50 \cdot 200 \cdot 200) = 5000$.

Becsülhetjük az összekapcsolás eredményében az egyes attribútumokhoz tartozó értékhalmazok méretét is. Egy-egy ilyen becsült értéket úgy kapunk, hogy vesszük a különböző relációkban – ahol az attribútum megtalálható – az attribútumhoz tartozó értékszámlálók legkisebbikét. Az a, b, c, d és e attribútumok esetében ezek a számok sorra a következők: 100, 20, 100, 400 és 500.

A két előfeltevésünkből adódóan a becslés fent megadott szabálya rendelkezik egy meglepő és kellemes tulajdonsággal, amely teljes indukcióval be is bizonyítható:

- Nem számít, hogy egy n relációt magában foglaló természetes összekapcsolást hogyan csoportosítunk és rendezünk, a becslésre vonatkozó szabályokat az egyes összekapcsolásokra egyenként alkalmazva ugyanazt a becslést kapjuk az eredmény méretére. Ez a becslés továbbá megegyezik azzal, amit akkor kapunk, ha az n reláció összekapcsolására vonatkozó szabályt alkalmazzuk.

Két korábbi példa szemléltette ezt a tulajdonságot, ahol három relációt kapcsoltunk össze háromféle csoportosításban, beleértve azt is, amikor az „összekapcsolások” egyike valójában szorzat volt.

Egyéb műveletek méretének becslése

Láttunk két műveletet, ahol az eredményül kapott sorok száma pontos formulával leírható:

1. A vetítés nem változtatja meg egy relációban szereplő sorok számát.
2. A szorzat olyan eredményt állít elő, amelyben a sorok száma egyenlő az argumentum relációkban lévő sorok számának szorzatával.

Két további műveletre (a kiválasztásra és az összekapcsolásra) elég jó becslési technikákat dolgoztunk ki. A fennmaradó műveletek esetében azonban nem könnyű az eredmény méretének meghatározása. Sorra vesszük a többi relációalgebrai operátort is, és javaslatokat teszünk arra, hogy ez a becslés hogyan végezhető el.

Egyesítés

Egy multihalmaz alapú egyesítés mérete pontosan az argumentumok méretének összegével egyenlő. Halmaz alapú egyesítés esetén a méret lehet olyan nagy, mint az argumentumok méretének összege, vagy olyan kicsi, mint a két méret közül a nagyobb. Célszerű e kettő átlágát venni becslésnek, ami ugyanaz, mint a nagyobb méret és a kisebb méret felének az összege.

Metszet

Az eredmény lehet üres, de lehet annyi sora is, mint a két argumentum közül a kisebbnek, függetlenül attól, hogy halmaz vagy multihalmaz alapú metszetről van szó. Az egyik lehetséges megközelítés most is az, hogy a szélsőségek átlagát vesszük, ami a kisebb felét jelenti.

Egy másik lehetőség, hogy felismerjük azt, hogy a metszet a természetes összekapcsolás egy speciális esete, így az arra vonatkozó szabályokat használjuk. Halmaz alapú metszet esetén ezek a formulák garantáltan olyan eredményt adnak, ami nem nagyobb, mint a két reláció közül a kisebb (ha az argumentum relációk is halmazok). Multihalmaz alapú metszet esetén azonban előfordulhatnak „rendellenességek", amikor a becslés nagyobb, mint bármelyik argumentum. Nézzük például az $R(a,b) \cap_B S(a,b)$ metszetet, ahol R a (0, 1) sor két példányából áll, S pedig ugyanennek a sornak három példányából áll. Ekkor $V(R,a) = V(S,a) = V(R,b) = V(S,b) = 1$, $T(R) = 2$ és $T(S) = 3$. Az összekapcsolásra vonatkozó szabály alapján a becslés $2 \cdot 3 / (1 \cdot 1) = 6$, de az eredményben nyilvánvalóan nem lehet több, mint $\min(T(R), T(S)) = 2$ sor.

Különbség

Amikor az $R - S$ különbséget vesszük, akkor az eredményben kapott sorok száma $T(R)$ és $T(R) - T(S)$ között lehet. Becslésként célszerű ezek átlagát tekinteni: $T(R) - T(S)/2$.

Ismétlődések megszüntetése

Ha $R(a_1, a_2, \ldots, a_n)$ egy reláció, akkor $\delta(R)$ mérete $V(R, [a_1, a_2, \ldots, a_n])$. Sokszor azonban nem rendelkezünk ezzel a statisztikai értékkel, ezért közelíteni kell. Mint szélsőségek, $\delta(R)$ mérete megegyezhet R méretével (ha nincsenek ismétlődések), vagy lehet 1 (ha R minden sora ugyanaz). (Sőt, lehet 0 is, ha R üres, de ez elég ritka eset.) Egy másik felső korlát a $\delta(R)$-ben lévő sorok számára az elképzelhető különböző sorok száma, ami a $V(R,a_i)$-k szorzata, ahol $i = 1, 2, \ldots, n$. Ez a szám lehet kisebb, mint $T(\delta(R))$ más becslései. Sok jó szabály létezik $T(\delta(R))$ becslésére. Az egyik elfogadható az, hogy $T(R)/2$ és az összes $V(R,a_i)$ szorzata közül vesszük a kisebbiket.

Csoportosítás és összesítés

Tegyük fel, hogy van egy $\gamma_L(R)$ kifejezésünk, és e kifejezés eredményének méretére kell becslést adnunk. Ha rendelkezünk a $V(R, [g_1, g_2, \ldots, g_k])$ statisztikával, ahol a g_i-k az L-ben szereplő csoportosító attribútumok, akkor az lesz a válaszunk. A statisztika azonban esetleg nem elérhető, így szükségünk van egy másik módszerre, amivel a $\gamma_L(R)$ méretét becsülhetjük. A $\gamma_L(R)$ sorainak száma megegyezik a csoportok számával. Az eredményben lehet, hogy egy csoport lesz, de lehet olyan sok csoport is, mint ahány sor van R-ben. A δ-hoz hasonlóan a csoportok számára a $V(R,g_i)$-k szorzatával is adhatunk felső korlátot, ahol $i = 1, 2, \ldots, k$. Újfent azt a becslést célszerű választani, ami veszi a $T(R)/2$ és e szorzat közül a kisebbiket.

Bevezetés a költség alapú tervválasztásba

Akár egy logikai terv kiválasztásáról van szó, akár egy konkrét logikai tervhez tartozó fizikai tervek közül való választásról, a lekérdezésoptimalizálónak becslést kell végeznie bizonyos kifejezések kiértékelésének a költségére vonatkozóan. A következőkben a költség alapú tervválasztásban felmerülő kérdéseket tárgyaljuk, amelyek közül a legfontosabb és legnehezebb problémáról – több reláció összekapcsolási sorrendjének megválasztásáról – a következő részben lesz szó.

Továbbra is azzal a feltételezéssel élünk, hogy egy kifejezés kiértékelésének költségét a végrehajtott lemez I/O-műveletek száma jól közelíti. A lemez I/O-műveletek számát pedig a következők befolyásolják:

1. A lekérdezés megvalósítására kiválasztott konkrét logikai operátorok (relációalgebrai műveletek), amelyek a logikai lekérdezésterv megválasztásának pillanatában dőlnek el.
2. A közbülső relációk méretei, amelyek becslését az előző részben tárgyaltuk.
3. A logikai operátorok megvalósítására használt fizikai operátorok (például hogy egymenetes vagy kétmenetes összekapcsolást válassztunk, vagy hogy rendezünk vagy nem rendezünk egy adott relációt). Erről a kérdésről később lesz szó.
4. A hasonló műveletek sorrendje, különös tekintettel az összekapcsolásra, amit a következő rész tárgyal.
5. Az argumentumok átadásának módszere, vagyis ahogy az argumentumok egy adott fizikai operátortól a következőhöz kerülnek. Erről szintén később lesz szó.

Sok feladatot kell megoldanunk ahhoz, hogy hatékony költség alapú tervválasztást valósítsunk meg. Először azt nézzük meg, hogy hogyan nyerhetjük ki leghatékonyabban az adatbázisból a méretre vonatkozó paramétereket, amelyeket a közbülső relációk méretének becsléséhez használunk fel. Ezután újra elővesszük az algebrai szabályokat, hiszen a költség alapú elemzést a jó logikai terv megtalálásakor is alkalmazzuk. Végül a kiválasztott logikai tervből származtatható összes fizikai terv felsorolására vonatkozóan nézünk meg különböző megközelítéseket. Különösen fontosak a kiértékelendő tervek számának csökkentésére irányuló módszerek, amelyek ugyanakkor biztosítják azt is, hogy nem szűrik ki a legkisebb költségű tervet.

A méretre vonatkozó paraméterek becslése

Az előző részben megadott formulákat arra alapozva fogalmaztuk meg, hogy ismerünk bizonyos fontos paramétereket, mint a `T(R)`, ami egy `R` reláció sorainak számát jelenti, vagy a `V(R,a)`-t, ami az `R` reláció `a` attribútumában előforduló különböző értékek számát jelöli. Egy modern adatbázis-kezelő rendszer általában lehetővé teszi, hogy a felhasználó (de legalábbis a DBA) közvetlenül kérje ezeknek a statisztikáknak az összegyűjtését. Ezek a statisztikák ezután használhatók a további lekérdezésoptimalizálások során a műveletek költségének becslésére. Ha ezt követő adatbázis-módosítások hatására megváltoznak a statisztikai értékek, a változásokat a rendszer csak egy újabb statisztikát gyűjtő parancs után veszi figyelembe.

Ha végigolvasunk egy teljes `R` relációt, akkor nyilván megszámolható a benne lévő sorok száma (`T(R)`), és minden `a` attribútumára az általa felvett különböző értékek száma (`V(R,a)`). Azoknak a blokkoknak a számát, amelyekben `R` elfér (`B(R)`), úgy becsülhetjük, hogy vagy megszámoljuk a blokkok tényleges számát (ha `R` nyalábolt), vagy `T(R)`-t elosztjuk azon sorok számával, amelyek egy blokkban elférnek, illetve az egy blokkban elhelyezhető sorok átlagos számával, ha a sorok változó hosszúságúak. `B(R)` e két becslése nem feltétlenül ugyanaz, de rendszerint „elég közeliek” a költségek összehasonlítása szempontjából, csak arra kell vigyázni, hogy mindig ugyanazt a megközelítést válasszuk.

Egy adatbázis-kezelő rendszer egy adott attribútum értékeinek *hisztogramját* is ki tudja számítani. Ha `V(R,a)` nem túl nagy, akkor a hisztogram az `a` attribútum minden értékéhez tartalmazza azok előfordulásának számát vagy arányát. Ha `a`-nak nagyon sok értéke van, akkor az is egy lehetőség, hogy csak a leggyakoribb értékeket rögzítjük külön-külön, a többi értéket pedig csoportokba soroljuk. A hisztogramok legjellemzőbb típusai a következők:

1. *Egyenlő szélességű hisztogram* (frequency histogram): Választunk egy `w` szélességet, és egy v_0 konstanst. Meghatározzuk azoknak a soroknak a számát, amelyekben lévő `v` értékre $v_0 \leq v < v_0 + w$. Ugyanezt tesszük akkor is, amikor $v_0 + w \leq v < v_0 + 2w$, és így tovább. A v_0 érték lehet a legkisebb

lehetséges érték, vagy az aktuális minimum érték. Utóbbi esetben ha egy v_0-nál kisebb értékkel találkozunk, csökkentjük v_0 értékét w-vel (vagy $k \cdot w$-vel), és egy (vagy több) új számlálót adunk a hisztogramhoz.

2. *Egyenlő magasságú hisztogram* (height-balanced histogram): Ebben az esetben nem a szélességet rögzítjük, hanem a csoportok (buckets) számát, és úgy osztjuk fel az aktuális értéktartományt, hogy minden csoportba nagyjából azonos számú érték kerüljön.
3. *Leggyakoribb értékek felsorolása*: Felsoroljuk a leggyakoribb értékeket és a hozzájuk tartozó gyakoriságokat. Az összes többi értéket tekinthetjük egyetlen csoportba tartozónak, de megadhatjuk őket egyenlő szélességű vagy egyenlő magasságú hisztogrammal is.

Egy hisztogram használatának az az előnye, hogy az összekapcsolások méretére az előző részben leírt egyszerűsített módszereknél pontosabb becslést adhatunk. Ha az összekapcsolási attribútum valamely értéke mindkét reláció hisztogramjában közvetlenül megjelenik, akkor pontosan tudjuk, hogy az eredménynek hány sorában fog szerepelni ez az érték (egyenlőség alapú összekapcsolás esetén). Az összekapcsolási attribútum azon értékei esetén, amelyek nem jelennek meg közvetlenül valamelyik reláció hisztogramjában, az összekapcsolásra gyakorolt hatás az előző részben leírtaknak megfelelően becsülhető. Ha egyenlő szélességű hisztogramokat használunk úgy, hogy a két reláció összekapcsolási attribútumaira ugyanazokat a sávokat alkalmazzuk, akkor becsülhetjük az egymásnak megfelelő sávok összekapcsolásainak méreteit, majd ezeket összeadhatjuk. Az eredmény helyes lesz, mert csak az egymásnak megfelelő sávokba eső sorok kapcsolódhatnak. A későbbiekben nem fogunk hisztogramokat használni a becslésekben, de most nézzünk néhány példát hisztogram alapú becslésre.

Példa. Vegyünk olyan hisztogramokat, amelyek a három leggyakoribb értéket és a hozzájuk tartozó gyakoriságokat tartalmazzák, és a maradék értékeket egy csoportba sorolják. Tegyük fel, hogy az `R(a,b) * S(b,c)` összekapcsolás méretét szeretnénk becsülni. Az `R.b`-re vonatkozó hisztogram legyen az alábbi:

1: 200, 0: 150, 5: 100, egyéb: 550

Az `R` reláció 1000 sorából tehát 550-ben nem 1, 0 vagy 5 `b` értéke, és ezen egyéb értékek közül egyik sem fordul elő 100-nál többször.

Az `S.b`-re vonatkozó hisztogram legyen a következő:

0: 100, 1: 80, 2: 70, egyéb: 250

Tegyük fel továbbá, hogy `V(R,b)` = 14 és `V(S,b)` = 13. Ez azt jelenti, hogy az `R` reláció azon 550 sora, amelyekben `b` értéke ismeretlen, 11 érték között van elosztva, vagyis átlagosan 50 sor jut mindegyikre. Az `S` relációban az a 250 sor, amelyekben `b` értéke ismeretlen, 10 érték között van elosztva, azaz átlagosan 25 sor jut minden ilyen értékre.

A 0 és 1 értékek explicit módon szerepelnek mindkét hisztogramban, így kiszámolhatjuk, hogy ha összekapcsoljuk `R`-nek azt a 150 sorát, amelyekben `b` értéke 0, `S`-nek azzal a 100 sorával, amelyekben `b` értéke 0, akkor ez 15000 sort eredményez. Ha összekapcsoljuk `R`-nek azt a 200 sorát, amelyekben `b` értéke 1, `S`-nek azzal a 80 sorával, amelyekben `b` értéke 1, akkor ez további 16000 sort eredményez.

A maradék sorok összekapcsolásából származó eredménysorok számának becslése összetettebb. Továbbra is fenntartjuk azt a feltevést, hogy a kisebb értékhalmazzal rendelkező relációban (`S`) előforduló minden érték a másik reláció értékhalmazában is szerepel. A `b` attribútum 11 fennmaradó `S`-beli értéke közül az egyikről tudjuk, hogy 2, egy másikról pedig feltesszük, hogy 5, hiszen ez az egyik leggyakoribb érték `R`-ben. Becslésként úgy tekintjük, hogy a 2 `R`-ben 50-szer, az 5 `S`-ben pedig 25-ször fordul elő. Ezeket a becsléseket úgy kapjuk, hogy feltételezzük, hogy az adott érték a megfelelő reláció hisztogramjában említett „egyéb” értékek egyike. A 2 értékből adódó további sorok száma így $70 \cdot 50 = 3500$, az 5 értékből származó további sorok száma pedig $100 \cdot 25 = 2500$.

Végezetül van még 9 olyan érték, ami mindkét relációban szerepel, és az ezekre vonatkozó becslésünk az, hogy mindegyik 50-szer fordul elő `R`-ben, és 25-ször `S`-ben. Így mind a kilenc érték $50 \cdot 25 = 1250$ további sorral járul hozzá az eredményhez. A végső eredmény méretének becslése tehát:

$15000 + 16000 + 3500 + 2500 + 9 \cdot 1250 = 48250$ sor

Az előző részből vett egyszerűbb becslés, ami azon alapul, hogy mindegyik érték ugyanannyiszor fordul elő az egyes relációkban, $1000 \cdot 500 / 14 = 35714$ sor lenne.

Példa. A következő példában egyenlő szélességű hisztogramokat feltételezünk, és azt demonstráljuk, hogy hogyan befolyásolja egy összekapcsolás méretének becslését az, ha tudjuk, hogy a két reláció értékhalmazai majdnem diszjunktak.

A relációk legyenek a következők:

```
Jan(dátum, hőmérséklet)
Júl(dátum, hőmérséklet)
```

A lekérdezés pedig legyen az alábbi:

```
SELECT Jan.dátum, Júl.dátum FROM Jan, Júl
  WHERE Jan.hőmérséklet = Júl.hőmérséklet;
```

A január és július hónapoknak azokat a napjait keressük tehát, amikor ugyanaz volt a hőmérséklet. A lekérdezésterv az, hogy a `Jan` és `Júl` relációkat a `hőmérséklet` attribútumok egyenlősége alapján összekapcsoljuk (nem természetes összekapcsolás!), majd vetítünk a két `dátum` attribútumra.

A `Jan` és `Júl` relációkhoz tartozó, a hőmérséklet attribútumokra vonatkozó hisztogramok a következők:

Sáv	*Jan*	*Júl*
–21 – –16	40	0
–15 – –10	60	0
–9 – –4	80	0
–3 – 2	50	0
3 – 8	10	6
9 – 14	6	20
15 – 20	0	50
21 – 26	0	100
27 – 32	0	60
33 – 38	0	10

Ha mindkét összekapcsolási attribútumhoz egyenlő szélességű hisztogram tartozik ugyanazokkal a sávokkal (amelyek közül némelyek esetleg üresek az egyes relációkban), akkor az összekapcsolás méretét úgy becsülhetjük, hogy az egyes sávokra leszűkített összekapcsolások méreteit becsüljük külön-külön, majd az így kapott becsléseket összeadjuk.

Ha két megfelelő sávnak T_1, illetve T_2 sora van, és a sávba tartozó értékek száma `V`, akkor – az előző részben lefektetett elveket követve – az ezekre a sávokra vonatkozó összekapcsolás méretére kapott becslés: $T_1 \cdot T_2$ / `V`. A fenti hisztogramok esetében ezeknek a szorzatoknak a nagy része 0, mert T_1 és T_2 közül legalább az egyik 0. Csak a 3–8 és a 9–14 sávok azok, amelyekben sem T_1, sem T_2 nem 0. Mivel egy sáv szélessége 6, a 3–8 sáv $10 \cdot 6 / 6 = 10$, a 9–14 sáv pedig $6 \cdot 20 / 6 = 20$ sort eredményez.

Az összekapcsolás méretének becslése tehát: $10 + 20 = 30$ sor. Ha nem lennének hisztogramok, és csak annyit tudnánk, hogy mindegyik relációnak 246 sora van, amelyek 60 db -21 és 38 közé eső érték mentén oszlanak el, akkor az összekapcsolás méretének becslése $246 \cdot 246 / 60 = 1009$ sor lenne.

Statisztikák növekményes kiszámítása

A lekérdezésoptimalizálók az egyes statisztikák bizonyos időközönkénti kiszámítását részesítik előnyben, mert ezek a statisztikák nem szoktak rövid időn belül radikálisan megváltozni. Továbbá, amint már említettük, még a pontatlan statisztikák is hasznosak, ha azokat mindegyik tervre egyformán alkalmazzuk. Teljes relációk megvizsgálása azonban még „időnként" is drága dolog. A statisztikák újraszámításának egy alternatívája a *növekményes kiértékelés* (incremental evaluation). Ez a módszer karbantartja és aktualizálja a paraméterekre vonatkozó becsléseket minden alkalommal, amikor az adatbázis módosul. Íme néhány módja annak, ahogyan ezt a rendszer megteheti:

- `T(R)` karbantartásához a rendszer egyet mindig hozzáad, amikor egy sort beszúrnak `R`-be, illetve egyet kivon belőle, valahányszor törölnek onnan egy sort. A módszer hátránya, hogy módosítani kell a beszúrást és a törlést végző eljárásokat, és növeli ezen műveletek költségét, bár a többletköltség általában jelentéktelen.
- Ha `R` valamely attribútumához létezik B-fa-index, akkor `T(R)`-t becsülhetjük úgy is, hogy csak a B^+-fában lévő blokkokat számoljuk meg. Feltehetjük például, hogy minden blokk 3/4 részben van tele, és az egy blokkban elférő kulcsok és mutatók számát használhatjuk a levélelemek által mutatott sorok számának becslésére. A módszer hátránya, hogy kevésbé pontos, mint `T(R)` közvetlen megszámolása, viszont csak akkor igényel teendőt, amikor megváltozik a B-fa-index szerkezete, ami aránylag ritka a beszúrásokhoz és törlésekhez viszonyítva.
- Ha az `R` reláció `a` attribútumához létezik index, akkor `V(R,a)`-t pontosan karbantarthatjuk. Egy `R`-be történő beszúráskor mindenképpen meg kell találnunk `a`-nak az új sorban lévő értékét az indexben, és ekkor megállapítjuk, hogy létezett-e már sor ugyanezzel az értékkel. Ha nem, akkor a `V(R,a)` számlálóhoz hozzáadunk egyet. Amikor pedig törlünk egy sort `R`-ből, akkor ellenőrizzük, hogy az utolsó sort töröljük-e az adott értékkel, és ha igen, akkor csökkentjük eggyel `V(R,a)`-t. Ezen módszer használatához az indexkarbantartó eljárást kell módosítani.
- Ha tudjuk, hogy `a` kulcsa `R`-nek, akkor azt is tudjuk, hogy `V(R,a) = T(R)`, anélkül hogy az indexet vizsgálnánk, vagy `V(R,a)`-t közvetlenül karbantartanánk.
- Ha nincs index az `a` attribútumra, akkor a rendszer létrehozhat egy kezdetleges „indexet" azáltal, hogy fenntart egy adatstruktúrát (például egy B-fát) `a` értékeinek tárolására.
- Végül az is egy lehetőség a `V(R,a)` statisztika kiszámítására, hogy akkor adunk rá statisztikai becslést, amikor szükség van rá, mégpedig az adatok egy kis része mint minta alapján. Ez egy bonyolult számítás, és számos feltételtől függ, például attól, hogy `a` értékei milyen eloszlást mutatnak. A vezérfonal a következő: Megnézzük `R` egy kis mintáját, például a sorainak az 1%-át. Ha azt találjuk, hogy `a` legtöbb értéke különböző, akkor `V(R,a)` valószínűleg közel van `T(R)`-hez. Ha viszont azt tapasztaljuk, hogy az `a`-k között nagyon kevés különböző érték szerepel, akkor az a valószínű, hogy az aktuális relációban létező legtöbb értékkel találkoztunk.

Felmerülhet a kérdés, hogy miért becslünk méretet a lemez I/O-műveletek száma helyett. A legfontosabb indok az, hogy amikor a logikai lekérdezésterveket vizsgáljuk, még nem döntöttük el azt, hogy mely fizikai operátorokat használjuk az egyes relcióalgebrai műveletek megvalósításához, így nem lehetünk biztosak egy adott logikai terv végrehajtásához szükséges lemez I/O-műveletek számában. Követhetjük azonban azt a heurisztikát, miszerint az a terv lesz valószínűleg a legjobb terv, amelyikre a közbülső relációk méreteinek összege a legkisebb. Ezt a heurisztikát az támasztja alá, hogy minél kisebbek a relációk, annál kevesebb lemez I/O-műveletet igényel azok olvasása és írása, és annál hatékonyabbak a rájuk vonatkozó műveleteket megvalósító algoritmusok.

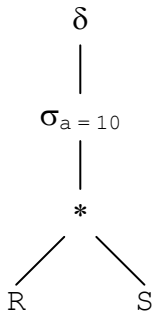
Egy lekérdezés igazi költségének becslését tényleg a lemez I/O-műveletek száma jelenti. Egy részletesebb elemzés figyelembe venné a CPU-időt is, egy még részletesebb elemzés pedig még a lemezfej mozgására

is kitérne, számításba véve az elért blokkok helyét a lemezen. A gyakorlatban azonban még a legegyszerűbb becslés (a közbülső relációk méretei) is elég jó, hiszen a lekérdezésoptimalizálónak csak lekérdezésterveket kell összehasonlítania, és nem kell pontos végrehajtási időt számolnia.

Logikai lekérdezéstervek költségének csökkentésére irányuló heurisztikák

A lekérdezésekre és alkérdésekre vonatkozó költségbecslések jól használhatók a lekérdezések heurisztikus átalakításai során. Az algebrai szabályok tárgyalásánál már megvizsgáltuk, hogy miként várható el, hogy bizonyos heurisztikák költségbecsléstől független alkalmazása szinte biztosan javítson egy logikai lekérdezésterv költségén. Jó példa erre a kiválasztások tologatása lefelé a fában. Vannak azonban a lekérdezésoptimalizálásnak más pontjai, ahol a költség becslése egy transzformáció előtt és után lehetővé teszi, hogy alkalmazzuk a transzformációt, ha várhatóan csökkenti a költséget, egyébként pedig elkerüljük. A végső logikai terv előállításakor például számba vehetünk számos opcionális transzformációt, és megnézhetjük az azok végrehajtása előtt és után előálló költségeket. Ezeket a kérdéseket szemlélteti a következő példa.

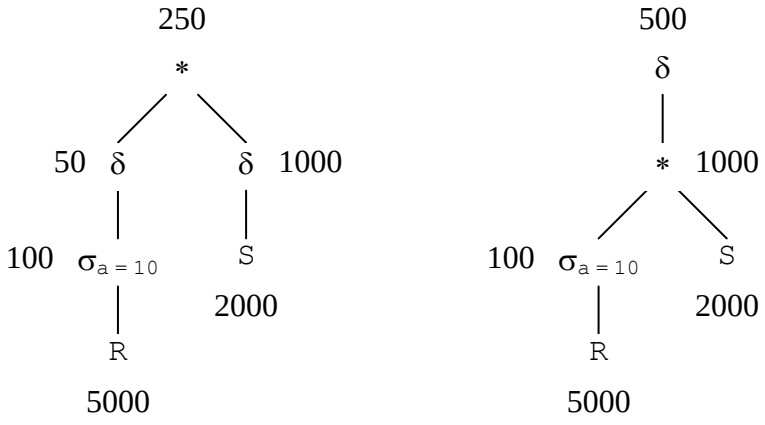
Példa. Tekintsük az alábbi kiindulási logikai tervet:



Legyenek az `R` és `S` relációkhoz tartozó statisztikák a következők:

R(a,b)	S(b,c)
T(R) = 5000	T(S) = 2000
V(R,a) = 50	
V(R,b) = 100	V(S,b) = 200
	V(S,c) = 100

A végső logikai lekérdezésterv előállításakor kitartunk amellett, hogy a kiválasztást levisszük a fában, ameddig csak lehet. Nem vagyunk viszont biztosak abban, hogy a δ operátor összekapcsolás alá történő levitelének van-e értelme vagy sem. A kiindulási tervből így két lekérdezéstervet generálunk, amelyek abban különböznek, hogy az ismétlődések megszüntetése az összekapcsolás előtt vagy után történik-e:



a) b)

Az a) tervben a δ-t levittük a fa mindkét ágán. Ha `R` és/vagy `S` nem tartalmaz ismétlődéseket, akkor a megfelelő ágon a δ elhagyható.

A kiválasztás méretének becslése: `T(R) / V(R,a)` = 5000 / 50 = 100. Az összekapcsolás méretének becslése: az argumentumok méreteinek szorzatát osztjuk `max(V(R,b), V(S,b))`-vel, azaz 200-zal. Azt viszont még nem tudjuk, hogy hogyan becsüljük az ismétlődésektől megszabadított relációk méretét.

Nézzük először a $\delta(\sigma_{a=10}(R))$ méretének becslését. Mivel a $\sigma_{a=10}(R)$-ben csak egyetlen értéke lehet `a`-nak és mintegy 100 értéke `b`-nek, és e reláció becsült mérete 100 sor, az előző részben megfogalmazott szabály azt mondja számunkra, hogy az egyes attribútumokhoz tartozó értékszámlálók szorzata nem korlátozó tényező, ezért a δ méretét a kiválasztásban szereplő sorok számának felével becsüljük, ami 50.

Nézzük most a b) ábrán lévő δ méretére vonatkozó becslést. Az összekapcsolás eredményében `a`-nak egyetlen értéke fordul elő, becslés alapján `b`-nek `min(V(R,b), V(S,b))` = 100 különböző értéke fordul elő, illetve becslés alapján `V(S,c)` = 100 különböző értéke szerepel `c`-nek. Vagyis ismét arra a következtetésre jutunk, hogy az értékszámlálók szorzata nem korlátozza a δ eredményének méretét, amelynek becslése tehát az összekapcsolás eredményében lévő sorok számának fele, azaz 500 sor lesz.

Ahhoz, hogy össze tudjuk hasonlítani az ábrán szereplő két tervet, összeadjuk a csomópontokhoz tartozó becsült méreteket, kivéve a gyökeret és a leveleket. A gyökeret és a leveleket azért hagyjuk ki, mert ezek a méretek nem függnek a választott tervtől. A költség, vagyis a közbülső relációkhoz tartozó becsült méretek összege, az a) terv esetében 100 + 50 + 1000 = 1150, míg a b) terv esetében 100 + 1000 = 1100. Arra a következtetésre jutunk tehát, hogy kis eltéréssel ugyan, de az ismétlődések megszüntetését a végére halasztva jobb tervet kapunk. Ellenkező következtetésre jutnánk akkor, ha `S`-ben `b`-nek kevesebb különböző értéke lenne. Ekkor nagyobb lenne az összekapcsolás mérete, ami megnövelné a b) terv költségét (a másik tervét nem, mert ott az összekapcsolás a gyökérben szerepel).

Érdekes jelenség, hogy a két fa gyökeréhez tartozó becslések különbözőek: 250 az egyikben, 500 a másikban. Mivel a becslés pontatlan tudomány, ilyen anomáliák előfordulnak. Valójában éppen az a ritka, hogy a konzisztenciára garanciát tudunk nyújtani, mint ahogy azt például az összkapcsolás méretének becslésénél láttuk.

Intuitíve azt mondhatnánk, hogy a b) terv becsült költsége magasabb, mert ha mind `R`-ben, mind `S`-ben vannak ismétlődések, akkor azok sokszorozódnak az összekapcsolás során. Azok a sorok például, amelyek háromszor szerepelnek `R`-ben és kétszer `S`-ben, hatszor fognak megjelenni az `R * S` összekapcsolás eredményében. A mi egyszerű formulánk, amellyel egy δ eredményének méretét becsüljük, nem számol azzal a lehetőséggel, hogy korábbi műveletek felerősíthetik az ismétlődések hatását.

Fizikai tervek felsorolásának lehetőségei

Most azt vizsgáljuk meg, hogy egy logikai lekérdezésterv fizikai lekérdezéstervvé történő konvertálása során hogyan használjuk a költségbecslést. A legalapvetőbb, *kimerítő* megközelítés szerint vesszük az előző rész elején felvázolt pontokra (összekapcsolások sorrendje, operátorok fizikai implementációja stb.) adott lehetséges válaszok összes kombinációját, majd az így kapott fizikai tervekhez egy-egy becsült költséget rendelünk, és a legkisebb költségű tervet választjuk.

Sok egyéb megközelítés is létezik azonban egy fizikai terv kiválasztására. A lehetséges fizikai tervek tartományának feltárására alapvetően kétféle módszer létezik:

- *Felülről lefelé*: Itt a gyökértől indulunk el, és haladunk lefelé a logikai lekérdezésterv fájában. A gyökérben található művelet minden lehetséges megvalósításához megnézzük az argumentum(ok) lehetséges kiértékeléseit, kiszámoljuk az egyes kombinációk költségét, és kiválasztjuk a legjobbat.
- *Alulról felfelé*: A logikai lekérdezésterv fájának minden részkifejezéséhez kiszámoljuk a részkifejezés lehetséges kiszámítási módjaihoz tartozó költségeket. Egy `E` részkifejezés kiértékelési lehetőségeit és költségeit úgy számítjuk ki, hogy vesszük az `E` részkifejezéseire vonatkozó lehetséges választásokat, és az összes lehetséges módon kombináljuk azokat az `E` gyökerében szereplő operátor lehetséges megvalósításaival.

Valójában nincs sok különbség a két megközelítés között, hiszen legrosszabb esetben mindkettő tekintetbe veszi a lekérdezésben szereplő összes operátor megvalósítási módjainak minden lehetséges kombinációját. Egy felülről lefelé módszer azonban esetleg lehetővé teszi, hogy elhagyjunk bizonyos választásokat, amelyeket alulról felfelé megközelítésben nem hagyhatnánk el. Kidolgoztak viszont olyan alulról felfelé stratégiákat is, amelyek jelentősen korlátozzák a választásokat, az elkövetkezőkben ilyenekre is láthatunk néhány példát.

Heurisztikus választás

Az is egy megoldás, hogy egy fizikai terv kiválasztására ugyanazt a megközelítést alkalmazzuk, mint amit általában egy logikai terv kiválasztására használunk: válasszunk heurisztikák alapján. A következő részben például egy mohó heurisztikát fogunk tárgyalni az összekapcsolási sorrendre vonatkozóan. Eszerint először azt a két relációt kapcsoljuk össze, amelyekre az eredmény becsült mérete a legkisebb, majd ugyanezt az elvet alkalmazzuk ismételten az így kapott reláció és a többi összekapcsolásra váró reláció összekapcsolása során. Sok egyéb alkalmazható heurisztika létezik, íme néhány a leggyakrabban használtak közül:

1. Ha a logikai tervben egy $\sigma_{a=c}(R)$ alakú kiválasztás szerepel, és az `R` tárolt relációnak létezik indexe az `a` attribútumra vonatkozóan, akkor csak az indexet nézzük végig azoknak az `R`-beli soroknak a megtalálására, amelyekben `a` értéke egyenlő `c`-vel.
2. Ha a kiválasztás tartalmaz egy `a = c` feltételt és más feltételeket is, akkor a kiválasztás megvalósítható egy indexes kereséssel, valamint egy azt követő, a kapott sorokra vonatkozó újabb kiválasztással, amit a *szűrés* (filter) fizikai szintű operátor képvisel.
3. Ha egy összekapcsolás valamely argumentumának van indexe az összekapcsolási attribútum(ok)ra vonatkozóan, akkor használjunk indexes összekapcsolást azzal a relációval a belső ciklusban.
4. Ha egy összekapcsolás egyik argumentuma rendezett az összekapcsolási attribútum(ok) szerint, akkor a rendezéses összekapcsolást részesítjük előnyben a tördelő összekapcsolással szemben, de nem feltétlenül az indexes összekapcsolással szemben, ha olyan is lehetséges.
5. Három vagy több reláció egyesítése vagy metszete során először a legkisebb relációkat csoportosítsuk.

Ág és korlát

Ebben a – gyakorlatban gyakran használt – megközelítésben azzal kezdjük, hogy valamilyen heurisztikát alkalmazva egy jó fizikai tervet keresünk a teljes logikai lekérdezéstervhez. Legyen ennek a tervnek a költsége C. Ezután, miközben részkifejezésekhez tartozó további terveket vizsgálunk, elvethetünk minden olyan tervet egy részkifejezés esetén, amelynek költsége nagyobb C-nél, hiszen egy ilyen – részkifejezéshez tartozó – terv nem lehet része egy olyan – a teljes lekérdezéshez tartozó – tervnek, amelytől azt várjuk, hogy jobb, mint amit már ismerünk. Ha egy olyan tervet állítunk elő a teljes lekérdezésre vonatkozóan, amelynek költsége kisebb, mint C, akkor a fizikai lekérdezéstervek tartományának további feltárása során C-t ennek a tervnek a költségével helyettesítjük.

E megközelítésnek egy nagy előnye, hogy megválaszthatjuk, mikor vetünk véget a keresésnek, és vesszük az addig talált legjobb tervet. Ha például a C költség kicsi, akkor még ha esetleg léteznek is sokkal jobb tervek, a megtalálásukra fordított idő meghaladhatja C-t, ezért nincs értelme a keresést tovább folytatni. Ha azonban C nagy, akkor bölcs dolog még arra időt fordítani, hogy egy jobb tervet keressünk.

Hegymászás

Ebben a módszerben, ahol valójában egy „völgyet” keresünk a fizikai tervek és költségeik tartományában, egy heurisztikusan kiválasztott fizikai tervvel kezdünk. A terven ezután kis változtatásokat hajtunk végre, például egy operátorhoz adott módszert egy másikkal helyettesítünk, vagy átrendezzük az összekapcsolásokat, hogy kisebb költségű „közeli” terveket találjunk. Amikor elérünk egy olyan tervhez, amelynek semmilyen kis változtatása nem eredményez alacsonyabb költségű tervet, a tervet kikiáltjuk a választott fizikai lekérdezéstervnek.

Dinamikus programozás

Az alulról felfelé módszernek létezik egy nyilvánvaló egyszerűsítése, mégpedig amikor egy nagyobb részkifejezés kiszámítása során csak a legjobb tervet vesszük figyelembe annak minden részkifejezése esetén. Amint haladunk felfelé a fában, megvizsgáljuk az egyes csomópontok lehetséges megvalósításait, mindegyik részkifejezéshez a legjobb tervet feltételezve. Ez a módszer nem biztos, hogy a legjobb tervet eredményezi, habár ez gyakran megtörténik. A következő részben kimerítően tárgyaljuk ezt a megközelítést.

Selinger-féle optimalizálás (System-R-módszer)

Ez a megközelítés a dinamikus programozás módszerén javít annyiban, hogy az egyes részkifejezésekhez nem csak a legalacsonyabb költségű tervet tartja meg, hanem bizonyos egyéb terveket is, amelyek magasabb költségűek ugyan, de amelyek a kifejezésfa feljebb eső részeinél jól kihasználható rendezettséggel rendelkező eredményt állítanak elő. Ilyen – számunkra érdekes – rendezettségre példa, amikor a részkifejezés eredménye az alábbiak valamelyike szerint van rendezve:

1. Egy gyökérben elhelyezkedő rendezés (τ) operátorban megadott attribútum(ok).
2. Egy későbbi csoportosítás (γ) operátor csoportosítási attribútuma(i).
3. Egy későbbi összekapcsolás összekapcsolási attribútuma(i).

Ha egy terv költségét a közbülső relációk méretei összegének vesszük, akkor egy argumentum rendezettsége nem látszik előnyösnek. Ha azonban az ennél pontosabb becslést (a lemez I/O-műveletek számát) használjuk költségként, akkor egy argumentum rendezettségének előnye világossá válik, hiszen használhatjuk a rendezettségen alapuló algoritmusok valamelyikét, és a már rendezett argumentumra megtakaríthatjuk az első menetet, a rendezést.

Összekapcsolások sorrendjének megválasztása

Ebben a részben a költség alapú optimalizálás egyik kritikus problémájára összpontosítunk, ami nem más, mint az összekapcsolási sorrend megválasztása három vagy több reláció (természetes) összekapcsolása esetén. Hasonló kérdések megfogalmazhatók más bináris műveletek (például egyesítés vagy metszet) kapcsán is, de ezek a műveletek nem annyira jelentősek a gyakorlatban, mivel végrehajtásuk általában kevesebb időt igényel, mint az összekapcsolásé, és ritkábban is szerepelnek kettőnél több argumentummal.

Összekapcsolások bal és jobb oldali argumentumainak jelentősége

Egy összekapcsolás sorrendbe állításakor tudnunk kell, hogy az ismert összekapcsolási módszerek közül sok aszimmetrikus abban az értelemben, hogy a két argumentum reláció szerepe különböző, és az összekapcsolás költsége függ attól, hogy melyik szerepet melyik reláció játssza. Az *egymenetes összekapcsolási algoritmus* például beolvassa az egyik relációt (lehetőleg a kisebbet) a központi memóriába, létrehoz egy adatstruktúrát, hogy elősegítse a másik relációból álló sorok illesztését, majd beolvassa a másik relációt blokkonként, és annak sorait összekapcsolja a memóriában tárolt sorokkal. Ennél az algoritmusnál a bal oldali argumentumot tekintjük annak a (kisebb) relációnak, amelynek sorait a központi memória adatszerkezetében tárolni fogunk (ezt a relációt nevezzük *építő relációnak* – building relation), míg az összekapcsolás jobb oldali argumentumát (a *vizsgáló relációt* – probing relation) fogjuk blokkonként beolvasni, és sorait illeszteni az építő reláció soraival.

Az egymenetes összekapcsolási algoritmuson kívül az argumentumokat megkülönbözteti a *beágyazott ciklusú összekapcsolás* (itt azt feltételezzük, hogy a bal oldali argumentum a külső ciklus relációja) és az *indexes összekapcsolás* is (itt pedig azt feltételezzük, hogy a jobb oldali argumentum rendelkezik az indexszel).

Nézzük meg kicsit részletesebben az *egymenetes összekapcsolás* (one-pass join) és a *beágyazott ciklusú összekapcsolás* (nested-loop join) algoritmusát!

Az egymenetes összekapcsolás algoritmusa

Tegyük fel, hogy az `R(X,Y)` és az `S(Y,Z)` relációkat szeretnénk összekapcsolni, ahol `Y` jelöli `R` és `S` közös attribútumainak halmazát, `X` jelöli `R` összes olyan attribútumának halmazát, amelyek nincsenek benne `S` sémájában, `Z` pedig `S` azon attribútumainak halmazát, amelyek nincsenek benne `R` sémájában. Feltesszük, hogy `R` a kisebbik reláció. A természetes összekapcsolás kiszámítását a következőképpen végezzük el:

1. Olvassuk be `R` összes sorát, és alkossunk belőlük egy olyan memóriabeli adatszerkezetet, amelyben `Y` a keresési kulcs. Jó példa ilyen adatszerkezetre egy tördelőtáblázat vagy egy kiegyensúlyozott keresőfa. Használjunk erre a célra `M-1` memóriablokkot, ahol `M` az összekapcsolásra szánt memóriapufferek száma.
2. Olvassuk be `S` minden egyes blokkját a fennmaradó egyetlen memóriapufferbe. `S` minden egyes `t` sorára keressük meg a keresési struktúrában `R` azon sorait, amelyek megegyeznek `t`-vel `Y` összes attribútumán. `R` minden ilyen sorára alkossunk egy sort a `t`-vel való összekapcsolással, majd az eredményt írjuk a kimenetre.

Az algoritmusnak `B(R) + B(S)` lemez I/O-műveletre van szüksége az operandusok beolvasásához. Az egymenetes összekapcsolás csak akkor alkalmazható, ha $B(R) \leq M-1$. (A keresési struktúra által igénybe vett tárhely elhanyagolható.)

A beágyazott ciklusú összekapcsolás algoritmusa

Ezt az algoritmust szokás „másfél menetes” összekapcsolásnak is hívni, mivel a két argumentum egyikéhez tartozó sorokat csak egyszer olvassuk be, a másik argumentum sorait viszont többször. Beágyazott ciklusú összekapcsolás alkalmazható bármekkora méretű relációk esetén, nem szükséges, hogy az egyik reláció elférjen a memóriában.

Lássuk először a *sor alapú beágyazott ciklusú összekapcsolás* algoritmusát:

```
FOR R minden egyes r sorára DO
  FOR S minden egyes s sorára DO
    IF r és s összekapcsolható egy t sorrá THEN
      t kiírása;
```

Ha nem figyelünk arra, hogy hogyan pufferelјük R és S blokkjait, akkor a fenti algoritmus akár $T(R) \cdot T(S)$ számú lemez I/O-műveletet is igényelhet. Sok esetben az algoritmus módosításával sokkal alacsonyabb költség is elérhető. Ilyen eset például az, amikor az S-beli összekapcsoló attribútum(ok)ra létezik index, amelynek segítségével megkereshetjük S sorai között azokat, amelyek megfelelnek R egy adott sorának, és ilyenkor nem kell a teljes S relációt beolvasnunk. Ez az *index alapú összekapcsolás*.

Egy másik javítási lehetőség figyelembe veszi azt, hogy R és S sorai miként vannak megosztva a blokkok között, és a lehető legtöbb memóriát használja fel a lemez I/O-műveletek számának csökkentésére, amikor a belső cikluson végighalad. Ez a *blokk alapú beágyazott cilkusú összekapcsolás*, amelynek algoritmusa a következő (feltételezzük, hogy $M < B(R) \leq B(S)$, azaz egyik reláció sem fér be teljesen a memóriába):

```
FOR R minden M-1 blokkból álló darabjára DO BEGIN
  olvassuk be ezeket a blokkokat a memóriapufferekbe;
  a sorokat szervezzük egy keresési struktúrába,
    amelynek kulcsa Y;
  FOR S minden egyes b blokkjára DO BEGIN
    olvassuk be b-t a memóriába;
    FOR b minden t sorára DO BEGIN
      keressük meg R azon sorait a memóriában,
        amelyek kapcsolódnak t-vel;
      írjuk ki e sorok t-vel való összekapcsolását;
    END;
  END;
END;
```

Amint az látható, mindkét argumentum relációnál blokkonkénti hozzáférést valósítunk meg, és a lehető legtöbb memóriát használjuk az R reláció sorainak tárolására. A blokkonkénti hozzáférés biztosítja, hogy amikor a belső ciklusban végigmegyünk az S reláció sorain, akkor a lehető legkevesebb lemez I/O-műveletet használjuk fel. Azáltal, hogy a lehető legtöbb ($M-1$) memóriapuffert szánjuk R sorainak a tárolására, lehetőségünk nyílik arra, hogy S minden egyes beolvasott sorát ne csak egy R-beli sorral kapcsoljuk össze, hanem annyival, amennyi csak elfér a memóriában.

Példa. Legyen $B(R) = 500$, $B(S) = 1000$ és $M = 101$! 100 darab memóriapuffert fogunk használni R 100 blokkos darabokban történő pufferelésére, a külső ciklust tehát ötször kell végrehajtani. Minden iteráció alkalmával 100 lemez I/O-művelettel olvassuk be R egy darabját, majd S-et teljes egészében be kell olvasnunk a belső ciklusban, amihez 1000 lemez I/O-műveletre van szükség. Így az összes lemez I/O-műveletek száma 5500.

Ha felcserélnénk a két argumentumot, akkor az algoritmus valamivel több lemez I/O-műveletet használna fel. Ekkor 10 alkalommal hajtódna végre a külső ciklus, alkalmanként 600 lemez I/O-műveletet végezve,

azaz az összes lemez I/O-műveletek száma 6000 lenne. Általában igaz, hogy némi előnnyel jár, ha a külső ciklusban a kisebb reláción megyünk végig.

Általában véve a beágyazott ciklusú összekapcsolás nem a rendelkezésünkre álló leghatékonyabb összekapcsolási algoritmus (korai adatbázis-kezelő rendszerekben ez volt az egyetlen lehetőség), viszont nagy előnye, hogy kiválóan megvalósítható iterátorok segítségével. Az *iterátor* nem más, mint három függvény (`Open`, `GetNext` és `Close`) együttese (interfész), amely lehetővé teszi, hogy a felhasználó soronként férjen hozzá egy fizikai operátor eredményéhez. Ezeknek a függvényeknek (metódusoknak) több különböző megvalósítása (implementációja) létezik, attól függően, hogy melyik fizikai operátorra (osztályra) hívjuk meg őket. Feltételezzük, hogy minden fizikai operátorra létezik egy olyan osztály, amelynek objektumai olyan relációk, amelyeket az operátor előállíthat. Ha `R` egy ilyen reláció, akkor `R` iterátorának a függvényeire `Open(R)`, `GetNext(R)` és `Close(R)` módon hivatkozhatunk.

Példa. A legegyszerűbb iterátor az, amelyik a táblaátvizsgálás (TableScan) operátort valósítja meg:

```
PROCEDURE Open(R)
BEGIN
  b := R első blokkja;
  t := a b blokk első sora;
  Found := TRUE;
END;

FUNCTION GetNext(R)
BEGIN
  IF t túl van a b blokk utolsó során THEN BEGIN
    b := R következő blokkja;
    IF nincs több blokk THEN BEGIN
      Found := FALSE;
      RETURN;
    END
    ELSE
      t := a b blokk első sora;
  END;
  oldt := t;
  t := a b blokk következő sora;
  RETURN oldt;
END;

PROCEDURE Close(R)
BEGIN
END;
```

Az `R * S`-t megvalósító beágyazott ciklusú összekapcsolás fizikai operátor iterátora könnyen felépíthető `R` és `S` iterátoraiból a következőképpen:

```
PROCEDURE Open(R,S)
BEGIN
  Open(R);
  Open(S);
  r := GetNext(R);
END
```

```
FUNCTION GetNext(R,S)
BEGIN
  REPEAT
    s := GetNext(S);
    IF NOT Found THEN BEGIN
      Close(S);
      r := GetNext(R);
      IF NOT Found THEN
        RETURN;
      Open(S);
      s := GetNext(S);
    END;
  UNTIL (r és s összekapcsolható);
  RETURN r és s összekapcsolva;
END;

PROCEDURE Close(R,S)
BEGIN
  Close(R);
  Close(S);
END;
```

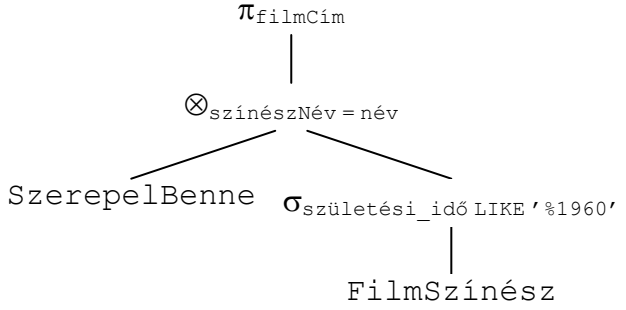
Összekapcsolási fák

Ha vesszük két reláció összekapcsolását, sorba kell rendezni az argumentumokat. Megegyezés alapján a kisebb becsült mérettel rendelkezőt tekintjük bal oldali argumentumnak. A fent említett algoritmusok mindegyike akkor működik a legjobban, ha a bal oldali argumentum a kisebb. Az egymenetes és a beágyazott ciklusú összekapcsolásban speciális szerep jut a kisebb relációnak (építő reláció, illetve külső ciklus), az indexes összekapcsolás pedig csak akkor ésszerű választás, ha az egyik reláció kicsi, a másiknak pedig van indexe. Általában jelentős különbség van az argumentumok méretében, ugyanis egy összekapcsolásokat tartalmazó lekérdezés nagyon gyakran tartalmaz legalább egy kiválasztást is, amely nagyban csökkentheti az egyik reláció becsült méretét.

Példa. Tekintsük a következő, már ismert lekérdezést:

```
SELECT filmCím FROM SzerepelBenne, FilmSzínész
  WHERE színészNév = név AND születési_idő LIKE '%1960';
```

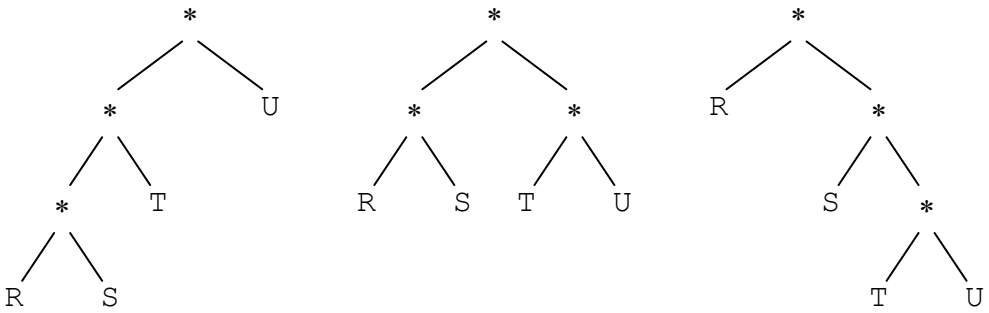
Az ehhez tartozó, előnyben részesített logikai lekérdezéstervet a következő ábra mutatja:



A `SzerepelBenne` és a `FilmSzínész` relációk méretére nem adtunk becslést, de feltételezhetjük, hogy az egy adott évben született színészek kiválasztása a `FilmSzínész` reláció sorainak körülbelül 1/50 részét állítja elő. Mivel általában több színész szerepel egy-egy filmben, a `SzerepelBenne`

reláció várhatóan nagyobb, mint a `FilmSzínész` reláció. Elmondható tehát, hogy az összekapcsolás második argumentuma sokkal kisebb, mint az első argumentum. Ebben a példában tehát meg kellene fordítani az argumentumok sorrendjét.

Két reláció esetén az összekapcsolási fára mindössze két lehetséges választás létezik. Gyorsan nő a lehetséges összekapcsolási fák száma, ha az összekapcsolás kettőnél több relációt foglal magában. Az alábbi ábra például három lehetséges szerkezetű fát mutat arra az esetre, amikor négy relációt kapcsolunk össze:



Mind a három fában azonban ugyanolyan sorrendben szerepel a négy reláció. Mivel számít az argumentumok sorrendje, és `n` relációt `n`! módon rendezhetünk, a levelek címkézési lehetőségeit tekintve mindegyik fa 4! = 24 különböző fát képvisel.

Bal-mély összekapcsolási fák

Egy bináris fa *bal-mély* (left-deep), ha minden nem levél csomópontjának jobb oldali gyermeke levélelem. Egy bináris fa *jobb-mély* (right-deep), ha minden nem levél csomópontjának bal oldali gyermeke levélelem. A se nem bal-mély, se nem jobb-mély fákat *bozótszerű* (bushy) fáknak nevezzük. A fenti példában a bal oldali fa bal-mély, a jobb oldali jobb-mély, a középső pedig bozótszerű.

Két előnye is van annak, ha lehetséges összekapcsolási sorrendként csak bal-mély fákat veszünk figyelembe:

1. Egy adott számú levéllel rendelkező lehetséges bal-mély fák száma nagy, de közel sem olyan nagy, mint az összes lehetséges fa száma. A nagyobb lekérdezésekhez tartozó lekérdezésterveket előbb felsorolhatjuk, ha a keresést a csak bal-mély összekapcsolási fákat tartalmazó tervekre korlátozzuk.
2. Az összekapcsolásokhoz használt bal-mély fák jól együttműködnek a szokásos összekapcsolási algoritmusokkal, különösen a beágyazott ciklusú és az egymenetes összekapcsolással. A bal-mély fákon alapuló felsorolási módszerek és ezek az algoritmusok várhatóan hatékonyabbak lesznek annál, mint ha ugyanezeket az algoritmusokat nem bal-mély fákkal használnánk.

Egy bal- vagy jobb-mély összekapcsolási fa „levelei” valójában lehetnek olyan részfák is, amelyek gyökerében szereplő operátor nem összekapcsolás. Az előző fejezet példájában szereplő összekapcsolási fa például tekinthető bal-mély összekapcsolási fának is (és természetesen jobb-mélynek is).

A bal-mély fák száma közel sem olyan mértékben nő, mint egy adott számú relációt magában foglaló sokargumentumos összekapcsoláshoz tartozó összes fa száma. `n` relációhoz egyetlen bal-mély faforma létezik, amelyhez `n`! módon rendelhetjük hozzá a relációkat. Ugyanennyi jobb-mély fa létezik. Az `n` relációhoz tartozó összes faforma számát az alábbi módon kapjuk meg:

$$T(1) = 1$$

$$T(n) = T(1)T(n-1) + T(2)T(n-2) + \ldots + T(n-1)T(1)$$

A második egyenlőséget az magyarázza, hogy a gyökér bal oldali részfájában szereplő levelek számát vehetjük egy tetszőleges 1 és `n – 1` közötti `i` egész számnak, és ezek a levelek `T(i)` különböző módon rendezhetők el. A jobb oldali részfában szereplő `n – i` számú levél pedig `T(n – i)` módon rendezhető el.

`T(n)` első néhány értéke: `T(1)` = 1, `T(2)` = 1, `T(3)` = 2, `T(4)` = 5, `T(5)` = 14, `T(6)` = 42. Az összes fa száma, amikor már a relációkat is hozzárendeljük a levelekhez, `T(n)` és `n!` szorzataként adódik. A 6 levelet tartalmazó megcímkézett levelű fák száma $42 \cdot 6! = 30240$, amelyek között van 720 bal-mély és másik 720 jobb-mély fa.

Nézzünk most két példát a bal-mély összekapcsolási fákkal kapcsolatban említett második előnyre, nevezetesen, hogy várhatóan hatékony terveket állítanak elő:

1. Ha egymenetes összekapcsolásokat használunk, és az építő reláció a bal oldalon van, akkor az egy időben szükséges memóriamennyiség feltehetően kisebb, mint ha egy jobb-mély vagy bozótszerű fát használnánk ugyanazon relációkra.
2. Ha iterátorokkal megvalósított beágyazott ciklusú összekapcsolásokat használunk, akkor elkerüljük azt, hogy valamely közbülső relációt egynél többször kelljen létrehozni.

Példa. Vegyük az előző ábrán látható bal-mély fát, és tegyük fel, hogy mind a három összekapcsoláshoz egy egyszerű egymenetes összekapcsolást fogunk használni, mégpedig úgy, hogy a bal oldali argumentum az építő reláció, vagyis azt tartjuk a központi memóriában. `R * S` kiszámításához az `R` relációt kell a központi memóriában tartanunk, és amint kiszámítottuk `R * S`-t, annak eredményét is a központi memóriában kell tartanunk. Tehát `B(R) + B(R * S)` számú központimemória-pufferre van szükségünk. Ha `R`-t vesszük a legkisebb relációnak, és egy kiválasztás eléggé kicsivé tette `R`-t, akkor valószínűleg nem lesz probléma, hogy rendelkezésre álljon ennyi puffer.

`R * S` kiszámítása után az eredményül kapott relációt össze kell kapcsolni `T`-vel. Az `R` tárolásához használt pufferekre már nincs tovább szükség, így azok újra felhasználhatók az `(R * S) * T` eredményének tárolásához. Amikor pedig ezt a relációt kapcsoljuk össze `U`-val, az `R * S` relációra már nincs tovább szükség, így az ehhez használt pufferek újra felhasználhatók a végső összekapcsolás eredményének tárolásához. Általánosan azt mondhatjuk, hogy egy egymenetes összekapcsolással kiszámított bal-mély összekapcsolási fa feltételezi, hogy legalább két ideiglenes reláció tárolásához szükséges hely mindig rendelkezésre áll a központi memóriában.

Most vizsgáljuk meg ugyanezt az algoritmust jobb-mély fa használatával. Elsőként az `R` relációt kell betölteni a központi memória puffereibe, mivel mindig a bal oldali argumentum az építő reláció. Ezután létre kell hozni az `S * (T * U)` relációt, és a gyökérben lévő összekapcsoláskor ezt kell használni vizsgáló relációként. `S * (T * U)` kiszámításához be kell vinni `S`-t pufferekbe, és ki kell számolni a `T * U` összekapcsolást mint a hozzá tartozó vizsgáló relációt. `T * U` kiszámításához viszont először pufferekbe kell tölteni `T`-t. Most tehát három relációt tárolunk egy időben a központi memóriában. Általánosan fogalmazva, ha egy `n` levéllel rendelkező jobb-mély összekapcsolási fát akarunk kiértékelni, akkor egyszerre `n – 1` relációt kell a központi memóriában tárolni.

Természetesen előfordulhat, hogy a `B(R) + B(S) + B(T)` összméret kisebb, mint a bal-mély fa kiszámítása során a két közbülső lépéshez szükséges helymennyiség, amelyek rendre `B(R) + B(R * S)` és `B(R * S) + B((R * S) * T)`. Ahogy azonban az előző fejezetben rámutattunk, a több összekapcsolást tartalmazó lekérdezéseknél gyakran van egy kis reláció, amely lehet a legbalra eső argumentum egy bal-mély fában. Ha `R` kicsi, akkor `R * S`-től azt várhatjuk, hogy lényegesen kisebb, mint `S`, az `(R * S) * T`-től pedig azt, hogy kisebb, mint `T`, ami csak további igazolását jelenti a bal-mély fák használatának.

Példa. Másik példaként tegyük fel, hogy az előző ábra négyes összekapcsolását beágyazott ciklusú összekapcsolásokkal szándékozzuk megvalósítani, és hogy az abban szereplő mindhárom összekapcsoláshoz egy-egy iterátort használunk. Az egyszerűség kedvéért azt is tegyük fel, hogy mind a

négy összekapcsolandó reláció tárolt reláció, nem pedig kifejezés. Ha a bal-mély fát használjuk, akkor a gyökérhez tartozó iterátor az `(R * S) * T` bal oldali argumentum egy a központi memória méretének megfelelő darabját kapja meg, majd ezt a darabot összekapcsolja a teljes `U`-val. Mivel `U` tárolt reláció, ezért csak végig kell olvasni, nem kell előállítani. Amikor megkapja, és a memóriába helyezi a bal oldali argumentum következő darabját, újra be kell olvasni `U`-t, de a beágyazott ciklusú összekapcsolásnál mindig szükség van erre az ismétlésre, ha mindkét argumentum nagy.

Ehhez hasonlóan az `(R * S) * T` összekapcsolás egy darabjának megszerzéséhez megkapjuk `R * S` egy darabját a memóriában, és átolvassuk `T`-t. `T` többszöri átolvasására most is szükség lehet, hacsak el nem kerülhető. Végül `R * S` egy darabjának megszerzése `R` egy darabjának beolvasását és `S`-sel történő összehasonlítását igényli, esetleg többször. Mindezen tevékenységek során azonban csak tárolt relációkat olvasunk többször, de ez az ismételt olvasás csak akkor jelent plusz költséget, ha a központi memória nem elég nagy ahhoz, hogy egy teljes reláció elférjen benne.

Hasonlítsuk most össze a bal-mély fához tartozó iterátorok viselkedését a jobb-mély fára vonatkozó iterátorok viselkedésével. A gyökérhez tartozó iterátor `R` egy darabjának beolvasásával kezd. Majd meg kell konstruálnia a teljes `S * (T * U)` relációt, és azt össze kell hasonlítania `R` megkapott darabjával. Amikor `R` következő darabját olvassuk be a memóriába, újra létre kell hozni az `S * (T * U)` relációt. Tehát `R` minden egyes darabjának beolvasásakor újra és újra elő kell állítani ugyanazt a relációt.

Természetesen megtehetnénk, hogy az `S * (T * U)` relációt egyszer létrehozzuk, és eltároljuk vagy a memóriában, vagy lemezen. Ha lemezen tároljuk el, akkor a bal-mély fához tartozó tervhez képest extra lemez I/O-műveleteket használunk, ha pedig a memóriában tároljuk, akkor belefutunk az előző példában felmerült problémába, a megnövekedett memóriaigénybe.

Dinamikus programozás az összekapcsolási sorrend és a csoportosítás megválasztására

Több reláció összekapcsolásakor a sorrend megválasztására három lehetőségünk van:

1. az összes lehetséges sorrendet figyelembe vesszük (ilyen a „mezítlábas” algoritmus);
2. a lehetséges sorrendek egy részhalmazát vesszük csak figyelembe (pl. dinamikus programozás);
3. egy heurisztikát használunk a sorrend kiválasztására (pl. mohó heurisztikát).

Most a felsorolás egy ésszerű megközelítését tárgyaljuk, amit *dinamikus programozásnak* hívnak. Ezt a módszert a lehetséges sorrendek bizonyos részhalmazainak megvizsgálására használhatjuk. Hamarosan megnézünk egy olyan elfogadható heurisztikát, amely egyetlen sorrend kiválasztására szolgál. A dinamikus programozás mögött az az alapötlet áll, hogy kitöltünk egy táblázatot a költségekről úgy, hogy csak a sikeres következtetéshez szükséges minimális információt őrizzük meg.

Tegyük fel, hogy végre akarjuk hajtani az $R_1 * R_2 * \ldots * R_n$ összekapcsolást. Ehhez létrehozunk egy olyan táblázatot, amelyben van egy bejegyzés az `n` reláció közül egyet vagy többet tartalmazó minden egyes részhalmazhoz. A táblázatban az alábbiakat helyezzük el:

1. Az adott relációk összekapcsolásának becsült mérete. Ennek meghatározásához a korábban bevezetett általános formulát használhatjuk.
2. Az adott relációk összekapcsolásához szükséges legkisebb költség. Ez lehet a közbülső relációk méreteinek összege, ahogy korábban is láttuk. Ha a lemez I/O-műveleteket vagy a futási idő más becslését használjuk költségként, akkor figyelembe kell venni az adott összekapcsoláshoz használt algoritmust is, mivel a különböző algoritmusok különböző költséggel járnak.
3. Az a kifejezés, amelyik a legkisebb költséget adja. Ez a kifejezés a kérdéses relációkat kapcsolja össze valamilyen csoportosításban. Korlátozhatjuk a keresést a bal-mély fákra, ekkor a kifejezés csak egy sorrendje a relációknak.

A táblázatot a részhalmaz méretére vonatkozó indukció alapján konstruáljuk meg. Két változat létezik, attól függően, hogy a fák összes lehetséges alakját figyelembe kívánjuk-e venni, vagy csak a bal-mély fákat (a különbség az indukciós lépésben lesz látható).

Alapeset: Az egyetlen R relációhoz tartozó bejegyzés R méretéből, 0 költségből és abból a formulából áll, ami maga R. Relációk egy $\{R_i, R_j\}$ párjára is könnyű kiszámítani a bejegyzést. A költség most is 0, mivel nincsenek közbülső relációk. A méret becslését a már jól ismert szabály adja meg: összeszorozzuk R_i és R_j méretét, majd R_i és R_j minden közös attribútuma esetén osztunk az attribútumhoz tartozó értékhalmazok méretei közül a nagyobbal. A formula pedig vagy $R_i * R_j$, vagy $R_j * R_i$. A rész elején bevezetett elvnek megfelelően R_i és R_j közül a kisebbet vesszük bal oldali argumentumnak.

Indukció: Ezután építjük tovább a táblázatot, kiszámítva a bejegyzéseket minden olyan részhalmazhoz, amelynek mérete nagyobb, mint 2, amíg el nem jutunk az egyetlen n méretű részhalmaz bejegyzéséhez. Egy-egy ilyen bejegyzés mondja meg, hogy melyik a legjobb módszer az ott szereplő összes reláció összekapcsolásának kiszámítására. Megadja továbbá ennek a módszernek a becsült költségét is, amire szükségünk van, amikor a későbbi bejegyzéseket számoljuk. Nézzük meg, hogyan számítunk ki egy k relációból álló H halmazhoz tartozó bejegyzést:

Ha csak bal-mély fákat szándékozunk figyelembe venni, akkor a H-hoz tartozó összekapcsolást úgy számítjuk ki, hogy a k relációból álló H minden egyes R relációja esetén először kiszámítjuk a $H - \{R\}$-hez tartozó összekapcsolást, majd ezt összekapcsoljuk R-rel. A H-hoz tartozó összekapcsolás költsége a $H - \{R\}$-hez tartozó költségnek és méretnek az összege lesz. A legkisebb költséget eredményező R-et választjuk. A H-hoz tartozó kifejezés bal oldali argumentuma a $H - \{R\}$-hez tartozó legjobb összekapcsolási kifejezés, a jobb oldali argumentuma pedig R lesz. A H-hoz tartozó méret az lesz, amit a sok reláció összekapcsolásánál használt formula ad.

Ha az összes fát figyelembe akarjuk venni, akkor a relációk egy H halmazához tartozó bejegyzés kiszámítása valamivel összetettebb. Meg kell vizsgálni az összes lehetséges módját a H halmaz H_1 és H_2 diszjunkt részhalmazokra történő particionálásának. Minden ilyen particionálás esetén vesszük az alábbiak összegét: a H_1-hez tartozó (legjobb) költség, a H_2-höz tartozó (legjobb) költség, H_1 mérete, H_2 mérete. Az egyes particionálások során előálló legkisebb ilyen összeget tekintjük a H-hoz tartozó költségnek. A H-hoz tartozó formula pedig a H_1-hez és a H_2-höz tartozó formulák összekapcsolása lesz.

Példa. Tekintsük az R, S, T és U relációk összekapcsolását. Az egyszerűség kedvéért feltételezzük, hogy mindegyiknek 1000 sora van. A relációk attribútumait és az egyes relációkban az attribútumokhoz tartozó értékhalmazok becsült méreteit a következő táblázat adja meg:

R(a,b)	S(b,c)	T(c,d)	U(d,a)
T(R) = 1000	T(S) = 1000	T(T) = 1000	T(U) = 1000
V(R,a) = 100			V(U,a) = 50
V(R,b) = 200	V(S,b) = 100		
	V(S,c) = 500	V(T,c) = 20	
		V(T,d) = 50	V(U,d) = 1000

Az egyelemű halmazokra vonatkozó méreteket, költségeket és a legjobb terveket a következő táblázat tartalmazza:

	{R}	{S}	{T}	{U}
Méret	1000	1000	1000	1000
Költség	0	0	0	0
Legjobb terv	R	S	T	U

Minden relációra a méret 1000, a költség 0, mivel nincsenek közbülső relációk, a legjobb (és egyetlen) kifejezés pedig maga a reláció.

Vegyük sorra most a relációpárokat. Mindegyikhez 0 költség tartozik, hiszen két reláció összekapcsolásakor még mindig nincsenek közbülső relációk. Két lehetséges terv létezik egy-egy párhoz, de mivel mindegyik reláció ugyanolyan méretű, mindegy, hogy melyiket választjuk. Az eredményül kapott relációk méreteit a szokásos képlet alapján számoljuk ki. A következő táblázat összegzi ezeket az eredményeket:

	{R,S}	{R,T}	{R,U}	{S,T}	{S,U}	{T,U}
Méret	5000	1M	10000	2000	1M	1000
Költség	0	0	0	0	0	0
Legjobb terv	R * S	R * T	R * U	S * T	S * U	T * U

Az olyan összekapcsolások mérete, amelyek valójában szorzatok, 1M = 1000000.

Nézzük most a négy reláció közül hármat magában foglaló összekapcsolásokhoz tartozó táblázatot. Három reláció összekapcsolásának kiszámításakor először ki kell választani közülük kettőt, amelyeket összekapcsolunk. Az eredmény méretének becslését a szokásos formulával számoljuk ki. Láttuk korábban, hogy ugyanazt a méretet kapjuk a különböző összekapcsolási sorrendek esetén.

Minden relációhármas esetén a költség az egyetlen közbülső relációnak (a kiválasztott két reláció összekapcsolásának) a mérete. Mivel azt akarjuk, hogy ez a költség a lehető legkisebb legyen, megvizsgáljuk a három relációból származó összes lehetséges relációpárt, és a legkisebb méretet eredményező párt választjuk.

A formula meghatározásakor először a két választott relációt csoportosítjuk egybe, melyek még lehetnek bal és jobb oldali argumentumok egyaránt. Tegyük fel, hogy csak a bal-mély fákat vesszük figyelembe, vagyis a kiválasztott két reláció összekapcsolását mindig bal oldali argumentumnak tekintjük. Ha megengednénk nem bal-mély fákat, akkor mindig az egyedül álló relációt választanánk bal oldali argumentumnak a példában (mivel az biztosan kisebb, mint bármelyik, két relációból álló összekapcsolás). A relációhármasokhoz tartozó táblázat a következő:

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
Méret	10000	50000	10000	2000
Költség	2000	5000	1000	1000
Legjobb terv	(S * T) * R	(R * S) * U	(T * U) * R	(T * U) * S

Példaként vizsgáljuk meg az {R,S,T} hármashoz tartozó számításokat. Meg kell néznünk az ezekből képezhető három pár mindegyikét. Ha az R * S-sel kezdünk, akkor a költség ennek a relációnak a mérete, ami 5000, amint ez a párokra vonatkozó táblázat alapján kiderül. Az R * T kezdés 1M-t ad költségként, ha pedig az S * T-vel kezdünk, akkor a költség 2000. A három lehetőség közül a legutóbbi jelenti a legkisebb költséget, így azt a tervet választjuk. Ezért szerepel a költségnél 2000, a legjobb tervnél pedig (S * T) * R.

Végül azt a helyzetet vizsgáljuk meg, amikor mind a négy relációt összekapcsoljuk. Ennek a relációnak a becsült mérete 100 sor, a költség most is a közbülső relációk összmérete. (Láttuk korábban, hogy tervek összehasonlításakor a végeredmény méretét soha nem vesszük figyelembe.) A mind a négy relációt magában foglaló összekapcsolás kiszámításának két általános módja van:

1. Kiválasztunk és a lehető legjobb módon összekapcsolunk hármat, majd az eredményt összekapcsoljuk a negyedikkel.
2. A négy relációt kétszer két párra osztjuk, összekapcsoljuk a párokat, majd összekapcsoljuk az eredményeket.

Ha csak bal-mély fákkal foglalkozunk, akkor a második típusú terveket kizárjuk, hiszen azok bozótszerű fákat eredményeznek. A következő táblázat összefoglalja az összekapcsolások hét lehetséges csoportosítását, amelyek az előző táblázatokban szereplő legjobb csoportosításokra épülnek:

Csoportosítás	Költség
`((S * T) * R) * U`	12000
`((R * S) * U) * T`	55000
`((T * U) * R) * S`	11000
`((T * U) * S) * R`	3000
`(T * U) * (R * S)`	6000
`(R * T) * (S * U)`	2000000
`(S * T) * (R * U)`	12000

Nézzük például az első formulát, amely azt ábrázolja, hogy először összekapcsoljuk az `R`, `S` és `T` relációkat, majd az így kapott eredményt összekapcsoljuk `U`-val. Az előző táblázatból tudjuk, hogy `R`, `S` és `T` összekapcsolásának a legjobb módja az, ha először `S`-et és `T`-t kapcsoljuk össze. Ennek a kifejezésnek a bal-mély alakját használtuk, és hogy a bal-mély forma megmaradjon, `U`-t jobb oldalról kapcsoltuk hozzá. Ha csak bal-mély fákat engedünk meg, akkor ez a kifejezés az egyetlen választási lehetőség. Ha megengednénk bozótszerű fákat is, akkor `U`-t bal oldalról kapcsolnánk, ugyanis kisebb, mint a másik három összekapcsolása. Az általunk kapott összekapcsolás költsége 12000, ami az `(S * T) * R` költségének és méretének az összege.

A táblázat utolsó három kifejezése további lehetőségeket ad, amennyiben bozótszerű fákat is figyelembe veszünk. Ezeket úgy kapjuk, hogy két relációpárt kapcsolunk össze. Az utolsó sor például azt a megoldást mutatja, hogy először elvégezzük az `R * U` és az `S * T` összekapcsolásokat, majd összekapcsoljuk azok eredményeit. Ennek a kifejezésnek a költsége a két pár méretének és költségének az összegeként adódik. Mindkét költség 0, a méretek pedig rendre 10000, illetve 2000. Mivel a kisebb relációt választjuk bal oldali argumentumnak, végül az `(S * T) * (R * U)` kifejezést kapjuk.

Azt látjuk ebben a példában, hogy a legkisebb költség a negyedik formulához tartozik, ezért ezt a kifejezést választjuk ki az összekapcsolás kiszámítására. Mivel ez egy bal-mély fa, ez a lekérdezésterv kerül kiválasztásra, függetlenül attól, hogy a mi dinamikus programozási stratégiánk mindenféle típusú tervet figyelembe vesz, vagy csak a bal-mély terveket. Ha megengedtünk volna nem bal-mély fákat is, akkor egy jobb-mély tervet kaptunk volna legjobb tervként, amelynek ugyanannyi a költsége, csak a formája más. Aszimmetrikus fizikai operátor használata esetén viszont nem lenne célszerű ezt a tervet választani, mert a tényleges költség sokkal nagyobb lenne, tehát ebben az esetben csak a bal-mély fákat kell figyelembe venni. A most használt költség fogalom ugyanis nem vesz figyelembe mást, csak a közbülső relációk méretét.

Dinamikus programozás részletesebb költségfüggvényekkel

Egy dinamikus programozási algoritmusban leegyszerűsíti a számításokat az, ha költségként relációméreteket használunk. Ennek az egyszerűsítésnek egy hátránya azonban az, hogy a számolás során az összekapcsolásoknak nem a tényleges költségeit vesszük figyelembe. Erre egy szélsőséges példa, amikor egy `R(a,b) * S(b,c)` összekapcsolásban az `R` relációnak egyetlen sora van, az `S` relációnak pedig van egy indexe a `b` összekapcsolási attribútumra, ekkor ugyanis az összekapcsolás szinte 0 időbe kerül. Másrészről viszont, ha `S`-nek nincs indexe, akkor végig kell olvasni, ami `B(S)` lemez I/O-műveletet igényel, még akkor is, ha `R` egyelemű. Egy olyan költség fogalom, ami csak `R`, `S` és `R * S` méretét veszi figyelembe, nem tud különbséget tenni e két eset között, vagyis a csoportosításban `R * S` használatának költségét vagy túlbecsüljük, vagy alulbecsüljük.

Nem nehéz azonban a dinamikus programozási algoritmust úgy módosítani, hogy az összekapcsolási algoritmusokat is számításba vegye. Először is a használt költség a lemez I/O-műveletek száma lesz, vagy valamilyen más, általunk előnyben részesített futásiidő-egység. A $H_1 * H_2$ költségének kiszámításakor

összeadjuk H_1 költségét, H_2 költségét és e két reláció összekapcsolásának legkisebb költségét, a legjobb rendelkezésre álló algoritmust használva. Mivel az utóbbi költség rendszerint függ H_1 és H_2 méretétől, ezeknek a méreteknek a becslését szintén ki kell számítani, ahogy ezt az előző példában tettük.

A dinamikus programozás egy még hatékonyabb változata a Selinger-féle optimalizálásra épül. Ekkor az egyes relációhalmazok esetén nem csak egy költséget tartunk meg, hanem többet. Amikor rendezett relációkat kell előállítani, a rendezéses összekapcsolás használatát figyelembe kell venni mint egy lehetőséget, ha viszont az eredmény rendezettsége nem értékes szempont, akkor a tördelő összekapcsolások minden esetben legalább olyan jók, mint a megfelelő rendezéses összekapcsolások.

Egy mohó algoritmus az összekapcsolási sorrend kiválasztására

Ahogy a fenti példa is mutatja, még a dinamikus programozás gondosan korlátozott keresése is exponenciális számú számításhoz vezet, az összekapcsolandó relációk számának függvényében. Öt vagy hat reláció összekapcsolásakor az optimális sorrend megtalálásához indokolt egy olyan alapos módszert használni, mint amilyen a dinamikus programozás vagy az „ág és korlát” algoritmus. Amikor azonban a relációk száma túlnő ezen, vagy ha az alapos kereséshez szükséges időt nem akarjuk ráfordítani, akkor használhatunk heurisztikus összekapcsolási sorrendet a lekérdezésoptimalizálóban.

A leggyakrabban választott heurisztika valamilyen *mohó* (greedy) heurisztika, ami azt jelenti, hogy egy adott ponton döntést hozunk az összekapcsolási sorrendre vonatkozóan, és soha nem lépünk vissza, hogy felülvizsgáljuk az egyszer már meghozott döntéseket. Most egy olyan mohó algoritmust vizsgálunk meg, amely csak bal-mély fákat választ ki. A „mohóság” alapja az az elv, hogy a köztes relációk méretét a lehető legkisebb értéken akarjuk tartani a fa minden szintjén.

Az algoritmus ezek alapján a következő: Kezdjük azzal a relációpárral, amelyek összekapcsolásának becsült mérete a legkisebb. Ezen relációk összekapcsolása lesz az *aktuális fa* (current tree). Ezután egy ciklusban az aktuális fában még nem szereplő relációk közül keressük meg azt, amelyiket az aktuális fával összekapcsolva a legkisebb becsült méretű relációt kapjuk. Az új aktuális fa bal oldali argumentuma a régi aktuális fa, jobb oldali argumentuma pedig a kiválasztott reláció lesz.

Példa. Alkalmazzuk a mohó algoritmust a dinamikus programozásnál látott példa relációira. Először megkeressük azt a relációpárt, amelyeknek az összekapcsolása a legkisebb méretű. A táblázatból kiderül, hogy ez a $T * U$ összekapcsolásra igaz, amelynek mérete 1000. Tehát $T * U$ lesz az aktuális fa. Ezután azt nézzük meg, hogy az R vagy az S relációt kapcsoljuk-e össze az aktuális fával következőként. Összehasonlítjuk tehát a $(T * U) * R$, illetve a $(T * U) * S$ relációk méretét. A táblázat szerint az utóbbi, amelynek mérete 2000, jobb, mint az első, amelynek mérete 10000. Az új aktuális fa ennek megfelelően a $(T * U) * S$ kifejezés lesz.

Nincs más lehetőségünk, mint hogy az utolsó lépésben az R relációt kapcsoljuk össze az aktuális fával. A teljes költség 3000 lesz, ami egyenlő a két közbülső reláció méretének összegével. Látható, hogy a mohó algoritmus ugyanazt a fát adja eredményül, mint amit a dinamikus programozási algoritmus. Léteznek azonban olyan példák, amikor a mohó algoritmus nem találja meg a legjobb megoldást, a dinamikus programozási algoritmus viszont garantáltan megtalálja a legjobbat (legalábbis ha nem veszünk figyelembe olyan heurisztikákat, mint például a rendezettség a Selinger-féle optimalizálásnál). Nem meglepő, hogy mindkét algoritmus eredménye a T és az U relációk összekapcsolásával kezd. Mivel $V(U,d) = T(U)$, a d attribútum nyilván kulcs U-ban. A többi attribútum viszont nem kulcs egyik relációban sem, ebből következik, hogy a legkisebb összekapcsolást úgy kapjuk, ha U-t kapcsoljuk össze a másik, d attribútumot is tartalmazó relációval, T-vel.

Futószalagosítás és materializáció

Egy lekérdezésterv végrehajtásának naiv módja az, hogy kialakítjuk a műveletek megfelelő sorrendjét azon egyszerű szabály szerint, hogy egy műveletet nem hajtunk végre addig, amíg a lekérdezéstervben alatta szereplő argumentumokat elő nem állítjuk, és minden művelet eredményét lemezen tároljuk mindaddig, ameddig egy másik műveletnek szüksége van rá. Ezt a stratégiát *materializációnak* (materialization) nevezzük, mivel minden közbülső reláció létrejön („testet ölt”) a lemezen.

A lekérdezéstervek végrehajtásának kifinomultabb és általában hatékonyabb módja az, amikor több művelet fut egyszerre. Egy adott művelet által előállított sorok közvetlenül átadódnak annak a műveletnek, amelyik használja azokat, anélkül hogy a sorokat bármikor is tárolnánk a lemezen. Ezt a módszert *futószalagosításnak* (pipelining) nevezzük, és általában iterátorok hálózatával valósítjuk meg, amelyek függvényei a megfelelő időben hívják egymást. A futószalagosításnak nyilván megvan a maga előnye, hiszen lemez I/O-műveleteket takarít meg, van azonban egy hátránya is: mivel egyszerre több műveletnek kell a központi memórián osztozni, a kevesebb rendelkezésre álló memória miatt megeshet, hogy magas lemez I/O-műveletszámú algoritmusokat kell választani.

Létezik egy közbülső megközelítés is a futószalagosítás és a materializáció között, amikor egy művelet teljes eredményét a központi memória puffereiben (tehát nem lemezen) tároljuk, mielőtt az azt felhasználó műveletnek átadódik. Az ilyen megoldást speciális futószalagosításnak tekintjük, ahol is első teendőként a felhasználó művelet elrendezi a memóriában a teljes relációt vagy annak egy nagy darabját. Ezt a módszert alkalmazhatjuk például egy olyan kiválasztásnál, amelynek eredménye továbbadódik egy egymenetes összekapcsolás építő argumentumaként.

Unáris műveletek futószalagosítása

Az unáris műveletek (kiválasztás és vetítés) kiváló jelöltek arra, hogy a futószalagosítást alkalmazzuk rajtuk. Mivel ezek a műveletek egyszerre egy sort dolgoznak fel, soha nincs szükség egy blokknál többre a bemenet számára, és egy blokknál többre a kimenet számára.

Egy futószalagosított unáris műveletet megvalósíthatunk iterátorok segítségével. A futószalagra kerülő eredményt felhasználó művelet meghívja a `GetNext()` függvényt, amikor egy újabb sorra van szüksége. Vetítésnél egyszer meg kell hívni a `GetNext()`-et az argumentum relációra, megfelelően vetíteni kell az adott sort, és visszaadni az eredményt a felhasználó művelet számára. Kiválasztásnál viszont előfordulhat, hogy többször kell meghívni a forrásra a `GetNext()`-et, amíg talál olyan sort, amely kielégíti a feltételt.

Bináris műveletek futószalagosítása

A bináris műveletekből származó eredményeket is lehet futószalagosítani. Egy puffert használunk arra a célra, hogy az eredményt a felhasználó műveletnek átadjuk, mégpedig blokkonként. Az eredmény kiszámításához és felhasználásához szükséges további pufferek száma azonban az eredmény méretétől és a lekérdezésben szereplő további relációk méreteitől függően változik. Mindezeket egy összetett példán keresztül szemléltetjük, de előtte tekintsük át röviden a *két- és többmenetes tördeléses összekapcsolás* (two-pass/multipass hash-based join) algoritmusát.

Kétmenetes tördeléses összekapcsolás

A tördeléses algoritmusok mögött meghúzódó alapötlet a következő: ha az adatok mennyisége túl nagy ahhoz, hogy azokat az elsődleges memória puffereiben tároljuk, akkor végezzünk *tördelést* (hashing) az argumentumok összes sorára egy megfelelő tördelőkulcs segítségével. Természetes összekapcsolásnál ezt

a kulcsot az összekapcsolási attribútumok alkotják. Ez azt jelenti, hogy mindkét argumentum relációhoz létrehozunk egy-egy „kulcstranszformációs táblázatot”, ahol a kulcs az összekapcsolási attribútumok összessége lesz, az érték egy teljes relációsor, a két hash-függvény pedig megegyezik. Azt mondjuk, hogy egy *kosárba* tartoznak a táblázatban azok az elemek (sorok), amelyekhez a hash-függvény ugyanazt az értéket (kosársorszámot) rendeli. A memóriabeli kulcstranszformációs táblázattól eltérően a kulcsokat (amelyek most azonosak is lehetnek két relációsorban) nem tároljuk, az értékeket (relációsorokat) pedig memóriapufferekben vagy lemezblokkokban tároljuk, és egy külön tömbben tartjuk nyilván, hogy hol helyezkednek el a memóriában, illetve a lemezen azok a blokkok, amelyek egy adott sorszámú kosárhoz tartozó relációsorokat tárolják.

Ezt követően úgy végezzük el az összekapcsolást, hogy egyszerre csak egy kosárpárral dolgozunk, ahol az egyik kosár a bal oldali, a másik pedig a jobb oldali argumentumhoz tartozó táblázat egy kosara, mégpedig az a kettő, amelyekben a sorokhoz a hash-függvény ugyanazt az értéket rendeli. Ezzel elérhetjük azt, hogy az operandusok méretét a kosarak számával arányos mértékben csökkentjük. Ha a kosarak száma `M`, akkor gyakorlatilag `M` db egymenetes összekapcsolást végzünk, amelyek mindegyike körülbelül `M`-ed akkora relációkat kapcsol össze, mint az eredeti relációk. A végeredményt a kapott `M` db reláció egyesítése adja. Ha `M` a rendelkezésre álló pufferek száma, akkor `M`–1 lehet a kosarak száma, és ezáltal egy `M`–1-es szorzóval növelhetjük az általunk kezelhető relációk méretét.

Most már csak azt kell megvizsgálnunk, miként particionálhatunk egy `R` relációt `M`–1 darab, nagyjából egyforma méretű kosárba, `M` puffer felhasználásával. Minden kosárhoz hozzárendelünk egy puffert. Az utolsó puffer fogadja `R` blokkjait, egyszerre egyet. A blokk minden egyes `t` sorát a `hash(t)` kosárba tördeljük, majd bemásoljuk a megfelelő pufferbe. Ha a szóban forgó puffer megtelt, akkor azt kiírjuk lemezre, majd ugyanahhoz a kosárhoz egy új blokkot inicializálunk (ugyanebben a pufferben). Amikor `R` összes blokkját beolvastuk, minden egyes nemüres kosarat kiírunk a lemezre. A tördelési algoritmus pszeudokódja a következő:

```
inicializáljunk M-1 kosarat M-1 üres puffer felhasználásával;
FOR R minden egyes b blokkjára DO BEGIN
  olvassuk be a b blokkot az M-edik pufferbe;
  FOR a b blokk minden egyes t sorára DO BEGIN
    IF a hash(t) kosárban nincs hely t számára THEN BEGIN
      másoljuk ki a puffert lemezre;
      inicializáljunk egy új üres blokkot a pufferben;
    END;
    másoljuk a t sort a hash(t) kosárhoz tartozó pufferbe;
  END;
END;
FOR minden egyes kosárra DO
  IF az adott kosárhoz tartozó puffer nem üres THEN
    írjuk ki lemezre az adott puffert;
```

Példa. Legyen `B(R)`=1000, `B(S)`=500, `M`=101! Ekkor megtehetjük, hogy mindkét relációt egyenként 100 kosárba tördeljük. Az átlagos kosárméret így `R` esetén 10, `S` esetén pedig 5 blokk. Mivel a kisebbik szám (5) jóval kisebb a rendelkezésre álló pufferek számánál, az egyes kosárpárok egymenetes összekapcsolása várhatóan nem fog akadályba ütközni (csak akkor, ha a hash-függvény nem elég jól véletlenszerűsít).

A lemez I/O-műveletek száma `R` és `S` kosarakba történő tördelése közben végzett beolvasáskor 1500, majd újabb 1500 I/O-művelet kell a kosarak lemezre írásához, végül ismét 1500 I/O-művelet szükséges az egyes kosárpárok memóriába történő beolvasásához, amikor az egymenetes összekapcsolásukat végezzük. A szükséges lemez I/O-műveletek száma tehát 4500.

Általánosítva kimondhatjuk, hogy a kétmenetes tördeléses összekapcsoláshoz $3 \cdot (B(R)+B(S))$ lemez I/O-művelet szükséges, és akkor alkalmazható, ha $\min(B(R),B(S)) \leq (M-1)^2$.

Többmenetes tördeléses összekapcsolás

Láthattuk, hogy két menet az összekapcsolás során elegendő, ha a kisebbik reláció nem „nagyon nagy". Könnyű látni azonban, hogy a kétmenetes algoritmusok olyan algoritmusokká általánosíthatók, amelyek tetszőleges méretű relációkra alkalmazhatók, szükség esetén több menetet használva. Ha a rendelkezésre álló memóriapufferek száma M, akkor osszuk fel a relációkat tördeléssel $M-1$ kosárba. Ezután végezzük el az összekapcsolást az egyes kosárpárokra egyenként, mintha azok teljes relációk lennének. A teljes relációkra alkalmazott művelet végeredménye megegyezik a kosárpárokra alkalmazott műveletek egyesítésével. Ezt a megközelítést rekurzívan a következőképpen írhatjuk le:

Alapeset: Ha bármelyik reláció befér $M-1$ pufferbe, akkor végezzük el az összekapcsolást úgy, hogy ezt a relációt beolvassuk a memóriába, majd a másik relációt blokkonként beolvassuk az M-edik pufferbe.

Indukció: Ha egyik reláció sem fér be a memóriába, akkor mindkettőt osszuk fel tördeléssel $M-1$ kosárba, majd a megfelelő kosarakból álló párokra végezzük el rekurzívan az összekapcsolást, végül egyesítsük a kosárpárok összekapcsolásából előálló eredményeket.

Könnyen belátható, hogy ha M a rendelkezésre álló memóriapufferek száma, akkor egy k menetes tördeléses összekapcsolás $(2k-1) \cdot (B(R)+B(S))$ lemez I/O-műveletet használ, és akkor alkalmazható, ha $\min(B(R),B(S)) \leq (M-1)^k$.

Példa a bináris műveletek futószalagosítására

Példa. Nézzünk fizikai lekérdezésterveket az alábbi kifejezéshez:

$(R(w,x) * S(x,y)) * U(y,z)$

A következő feltételezésekkel élünk:

1. $B(R) = 5000$, $B(S) = B(U) = 10000$.
2. Az $R * S$ közbülső eredmény által elfoglalt blokkok számát k-val jelöljük. k-t becsülhetnénk az R-ben és S-ben előforduló x értékek száma, valamint a (w,x,y) soroknak az R-beli (w,x) és az S-beli (x,y) sorokhoz viszonyított mérete alapján. Látni szeretnénk azonban, hogy mi történik, ha k változik, így nyitva hagyjuk ezt a konstanst.
3. Mindkét összekapcsolást tördelő összekapcsolással valósítjuk meg, mégpedig k-tól függően egymenetessel vagy kétmenetessel.
4. 101 puffer áll rendelkezésre. Ezt a számot mesterkélten alacsonyra vettük.

Nézzük először az $R * S$ összekapcsolást! Egyik reláció sem fér el a központi memóriában, ezért egy kétmenetes tördelő összekapcsolásra van szükségünk. Ahhoz, hogy a kisebb R reláció egyes kosarait 100 blokkra korlátozzuk, legalább 50 kosárra szükség van. Ha 100 kosarat használunk (és a hash-függvény „tökéletesen" véletlenszerűsít), akkor az $R * S$ tördelő összekapcsolás második menete 51 puffert használ, így szabadon marad 50 puffer, amit az $R * S$ eredményének U-val való összekapcsolásához lehet használni.

Tegyük fel, hogy $k \leq 49$, vagyis az $R * S$ összekapcsolás eredménye legfeljebb 49 blokkot foglal el! Ekkor az $R * S$ összekapcsolás eredményét 49 pufferbe futószalagosíthatjuk, a kikeresés céljából tördelőtáblába (vagy más keresési struktúrába) rendezhetjük, és marad egy blokk arra, hogy az U reláció egyes blokkjait sorra beolvassuk. A második összekapcsolást tehát egy egymenetes összekapcsolással végrehajthatjuk. A lemez I/O-műveletek száma:

a) R és S kétmenetes tördelő összekapcsolásának végrehajtásához: 3 · (5000 + 10000) = 45000.
b) Az (R * S) * U egymenetes (tördelő) összekapcsolásban U beolvasásához: 10000.

Ez összesen 55000 lemez I/O-művelet.

Most azt tegyük fel, hogy 49 < k ≤ 5000! R * S eredményét még mindig futószalagosíthatjuk, de most másik stratégiát kell alkalmazni: R * S eredményét U-val egy 50 kosaras kétmenetes összekapcsolással kapcsoljuk össze:

1. Mielőtt elkezdjük R és S összekapcsolását, U-t széttördeljük 50 darab 200 blokkos kosárba.
2. Következő lépésként elvégezzük R és S kétmenetes tördelő összekapcsolását 100 kosarat használva, csakúgy, mint az előbb. Most azonban amint előállítjuk R * S egy sorát, elhelyezzük azt a fennmaradó 50 puffer valamelyikében, amely pufferek R * S eredményének 50 kosárba történő tördeléséhez valók (ugyanazzal a hash-függvénnyel, amit U tördeléséhez is használtunk). Ezeket a puffereket lemezre írjuk, amikor megtelnek, ahogy az a kétmenetes tördelő összekapcsolások esetében szokásos.
3. Végezetül kosaranként összekapcsoljuk R * S eredményét U-val. Minthogy k ≤ 5000, R * S kosarai legfeljebb 100 blokk méretűek lesznek, ezért ez az összekapcsolás megvalósítható. Az a tény, hogy U kosarai 200 blokk méretűek, nem jelent problémát, hiszen a kosarak egymenetes összekapcsolásában R * S kosarait használjuk építő relációként, U kosarait pedig vizsgáló relációként.

Ehhez a futószalagosított összekapcsoláshoz tartozó lemez I/O-műveletek száma:

a) U beolvasásához és sorainak kosarakba történő írásához: 2 · 10000 = 20000.
b) Az R * S kétmenetes tördelő összekapcsolás végrehajtásához: 3 · (5000 + 10000) = 45000.
c) R * S kosarainak kiírásához: k.
d) R * S és U kosarainak beolvasásához a végső összekapcsolásban: k + 10000.

Az összköltség tehát 75000 + 2k.

Végül nézzük meg, hogy mi a helyzet, ha k > 5000! Most a rendelkezésre álló 50 pufferben nem végezhetünk kétmenetes összekapcsolást, ha R * S eredményét futószalagosítjuk. Használhatnánk hárommenetes összekapcsolást, de az mindkét argumentumnál blokkonként 2 extra lemez I/O-műveletet igényelne, ami 20000 + 2k-val több lemez I/O-műveletet jelentene, tehát az összköltség 95000 + 4k lenne. Jobban tesszük, ha inkább eltekintünk R * S futószalagosításától. Az összekapcsolások kiszámításának váza ebben az esetben a következőképpen néz ki:

1. Kiszámítjuk R * S-t egy kétmenetes tördelő összekapcsolást használva, és az eredményt eltároljuk lemezen.
2. R * S eredményét összekapcsoljuk U-val, szintén egy kétmenetes tördelő összekapcsolást használva. Mivel B(U) = 10000, a 100 blokktárolót használó kétmenetes tördelő összekapcsolás végrehajtható, függetlenül k nagyságától. Technikailag U-nak a megfelelő összekapcsolás bal oldali argumentumának kellene lennie, ha úgy döntünk, hogy U-t választjuk építő relációnak a tördelő összekapcsolásban.

Az ehhez az algoritmushoz szükséges lemez I/O-műveletek száma:

a) R és S kétmenetes tördelő összekapcsolásához: 3 · (5000 + 10000) = 45000.
b) R * S eredményének eltárolásához: k.
c) U és R * S kétmenetes tördelő összekapcsolásához: 3 · (10000 + k) = 30000 + 3k.

A teljes költség így 75000 + 4k, ami kevesebb, mint annak a költsége, ha az utolsó lépésben hárommenetes összekapcsolást használnánk.

A három algoritmus lényeges jellemzőit k függvényében a következő táblázat foglalja össze:

k *tartománya*	*Futószalagosítás vagy materializáció*	*Az utolsó összekapcsolás algoritmusa*	*Lemez I/O-műveletek száma*
$k \leq 49$	futószalagosítás	egymenetes	55000
$49 < k \leq 5000$	futószalagosítás	kétmenetes, 50 kosaras	$75000 + 2k$
$k > 5000$	materializáció	kétmenetes, 100 kosaras	$75000 + 4k$

Az Oracle lekérdezésoptimalizáló rendszere

A lekérdezésfordítás első lépései (elemzés és előfeldolgozás) elég hasonlóak az egyes adatbázis-kezelő rendszerekben. A nagy különbség a lekérdezések optimalizálásában, azaz a leghatékonyabb logikai és fizikai lekérdezésterv kiválasztásában van. Tekintsük most át nagyvonalakban azokat az eszközöket, amelyeket az Oracle alkalmaz a lekérdezésoptimalizálás során.

Az Oracle a lekérdezéstervet *végrehajtási tervnek* (execution plan) nevezi. A végrehajtási terv tartalmazza a lekérdezésben szereplő összes tábla *hozzáférési útját* (access path), az összekapcsolások sorrendjét és az összekapcsolások módját (az Oracle minden bináris operátort összekapcsolásnak tekint). A hozzáférési út azt mondja meg, hogy az adatokat milyen módon nyerjük ki az adatbázisból. Általában index alapú hozzáférést használunk, ha a táblából csak néhány sort olvasunk ki, míg a teljes átvizsgálás hatékonyabb, ha a tábla nagy részét feldolgozzuk.

Lehetőség van az egyes lekérdezésekhez előállított végrehajtási tervek lekérdezésére. SQL*Plus-ban ezt a `SET AUTOTRACE ON` utasítással tehetjük meg, amelynek hatására a lekérdezések végrehajtása után láthatjuk a hozzájuk tartozó végrehajtási tervet, valamint annak főbb jellemzőit (pl. az egyes lépések költségeit). A `SET AUTOTRACE ON TRACE EXPLAIN` utasítás hatására csak a végrehajtási tervet láthatjuk, a lekérdezések végrehajtása nélkül. Ha nincs hozzáférésünk az SQL*Plus-hoz, akkor a lekérdezést az `EXPLAIN PLAN FOR` előtaggal kell ellátnunk, hogy lekérdezhessük a végrehajtási tervet.

A lekérdezésoptimalizáló működési módjai

Az Oracle-ben a lekérdezésoptimalizáló két módban működhet: *normál módban* (normal mode) és *hangoló módban* (tuning mode). Normál módban az optimalizáló a legtöbb lekérdezés esetén elfogadható végrehajtási tervet generál. Szigorú időkorlátot használ (rendszerint a másodperc törtrészét), ami alatt elő kell állítania egy jó végrehajtási tervet. Hangoló módban az optimalizáló további elemzést végez, hogy a normál módban előállított terven lehet-e tovább javítani. Az Oracle ilyenkor *automatikus SQL-hangolásról* (Automatic SQL Tuning) beszél. Ez a folyamat akár percekig is eltarthat egyetlen lekérdezés esetén is, ezért ezt a módot csak összetett, sok adatmozgást igénylő lekérdezéseknél szokás használni. Az Automatic Database Diagnostic Monitor (ADDM) segíthet eldönteni, melyek ezek a lekérdezések.

Az automatikus SQL-hangolás lehetőségeit az SQL Tuning Advisor segédprogrammal lehet elérni, amelynek bemenete egy vagy több lekérdezés, kimenete pedig nem egy végrehajtási terv, hanem javaslatok egy sorozata, magyarázatokkal és a várható nyereségekkel. Ezek a javaslatok a különböző adatbázis-objektumokra vonatkozó statisztikák gyűjtésével, SQL profil létrehozásával, új indexek létrehozásával és a lekérdezés szerkezeti átalakításával kapcsolatosak. A felhasználó eldöntheti, hogy elfogadja-e ezeket az ajánlatokat a lekérdezések hangolásának befejezéséhez.

Az automatikus SQL-hangolás során a lekérdezésoptimalizáló négyféle elemzést hajt végre:

- *Statisztikaelemzés*: A lekérdezésoptimalizáló a különböző adatbázis-objektumokra vonatkozó statisztikákra alapozva állítja elő a végrehajtási terveket. Ha ezek a statisztikák elavultak vagy hiányoznak, az optimalizáló esetleg nem elég hatékony terveket fog készíteni. Automatikus SQL-

hangolás során az optimalizáló ellenőrzi a lekérdezés minden egyes objektumára a statisztikák meglétét, illetve frissességét, majd az alábbi kimenetek egyikével szolgál:

- Hiányzó vagy elavult statisztikákkal rendelkező objektumok esetén ajánlatot tesz ezen statisztikák előállítására. Mivel az optimalizáláshoz szükséges statisztikákat az Oracle automatikusan gyűjti és frissíti, ez a probléma csak akkor merül fel, ha ezt az automatikus statisztikagyűjtést kikapcsoltuk.
- Segédinformációkat szolgáltat, amelyek statisztikával nem rendelkező objektumok esetén maguk a statisztikák, elavult statisztikákkal rendelkező objektumok esetén pedig az azok kiigazításához szükséges információk. Ezeket a segédinformációkat az Oracle egy külön objektumban, az *SQL profilban* (SQL Profile) tárolja.

- *SQL profilok*: Normál módban a lekérdezésoptimalizáló néha pontatlan becsléseket ad egy lekérdezés egy attribútumára vonatkozóan a rendelkezésére álló információk hiánya miatt, amelynek következtében kevésbé hatékony végrehajtási terveket állíthat elő. Az automatikus SQL-hangolás ezt a problémát az SQL profilok segítségével hidalja át, amely a szóban forgó lekérdezésre vonatkozó statisztikákat tartalmazza. Ezeket a statisztikákat mintavételezéssel és részleges végrehajtással határozza meg, majd segítségükkel ellenőrzi, és ha kell, kiigazítja a normál módban előállított becsléseket. A fentieken kívül a lekérdezés korábbi végrehajtásai során szerzett információkra alapozva megfelelően beállítja az optimalizáló paramétereit. Ilyen paraméter például az `OPTIMIZER_MODE`, amely az optimalizáló alapértelmezett viselkedését állítja be. Ennek értéke lehet `ALL_ROWS` (ekkor költség alapú megközelítést használ a legkevesebb erőforrást igénylő terv megtalálására), `FIRST_ROWS_n` (ekkor költség alapú megközelítést használ az első `n` sor lehető leggyorsabb előállítására) vagy `FIRST_ROWS` (ekkor a költség alapú és a heurisztika alapú megközelítések keverékét használja az első néhány sor gyors előállítására). Az alapértelmezés az `ALL_ROWS`.

  Ennek az elemzésnek a kimenete egy ajánlás az SQL profil elfogadására. Ha elfogadjuk, a profil letárolásra kerül az adatszótárban, és ekkor a lekérdezésoptimalizáló normál módban is használja a szokásos adatbázis-statisztikákkal együtt a végrehajtási terv előállításához. Fontos, hogy egy profil egy konkrét lekérdezésre vonatkozik, és hogy a használata nem jelenti azt, hogy a lekérdezés minden egyes végrehajtása ugyanazon végrehajtási terv alapján történik. Ha megváltozik a táblák mérete, vagy létrehozunk, esetleg törlünk egy indexet, akkor a végrehajtási terv megváltozhat, míg az SQL profilban tárolt információk továbbra is érvényesek maradnak, még ha az adatok eloszlása vagy a lekérdezés hozzáférési útja meg is változik. Hosszú idő elteltével azonban ezek az információk is elévültté válhatnak, és ilyenkor újra kell őket generálni. Ezt úgy érhetjük el, hogy újra lefuttatjuk az automatikus SQL-hangolót a szóban forgó lekérdezésre.

- *A hozzáférési utak elemzése*: Egy lekérdezés végrehajtásának idejét jelentősen csökkenthetik az indexek, mivel elkerülhetjük velük egyes nagyméretű táblák teljes átvizsgálását. Az automatikus SQL-hangolás során az optimalizáló azt is megvizsgálja, hogy egy új index használata gyorsabb lekérdezés-végrehajtáshoz vezet-e. Ha talál ilyen indexet, javasolja annak létrehozását. Azt viszont nem elemzi, hogy az index létrehozása hogyan befolyásolja a teljes végrehajtási időt, ezért ilyenkor javasolja az SQLAccess Advisor segédprogram futtatását, amely megteszi ezt helyette, mielőtt javaslatot tesz.

- *A lekérdezés szerkezetének elemzése*: Az automatikus SQL-hangolás a lekérdezések általános szerkezeti problémáit is feltárja. Ezek lehetnek szintaktikai, szemantikai vagy tervezési problémák. Mindegyik esetben javaslatot tesz a lekérdezés átszervezésére. A javasolt változat hasonló, de nem ekvivalens az eredeti lekérdezéssel. Például javasolhatja a `UNION` operátor kicserélését `UNION ALL`-ra, vagy a `NOT IN` kicserélését `NOT EXISTS`-re. Az alkalmazásfejlesztő ezután eldöntheti, hogy ez a változtatás alkalmazható-e az adott szituációban vagy sem. Ha például nem állhatnak elő duplikált sorok, akkor a `UNION ALL` sokkal hatékonyabb, mint a `UNION` operátor. Az ilyen módosításokat csak alapos megfontolás után szabad megvalósítani.

A lekérdezésoptimalizáló működése

Az Orcale által feldolgozott minden egyes lekérdezés esetén a lekérdezésoptimalizáló a következő műveleteket hajtja végre:

- Kiértékeli a kifejezéseket és a feltételeket, amennyire csak lehetséges.
- Összetett (például korrelatív alkérdést vagy nézeteket tartalmazó) lekérdezések esetén átalakíthatja az eredeti lekérdezést egy ekvivalens összekapcsolássá (lekérdezésátírás).
- Megállapítja az optimalizálási célokat.
- A lekérdezés által elért minden egyes táblához kiválaszt egyet vagy többet a lehetséges hozzáférési utak közül a táblában tárolt adatok kinyeréséhez.
- Kettőnél több tábla összekapcsolását tartalmazó lekérdezés esetén meghatározza az összekapcsolási sorrendet.

A felhasználó befolyásolhatja az optimalizáló döntéseit az optimalizálás céljának megadásával és reprezentatív statisztikák gyűjtésével. Az optimalizálás célját alapértelmezésben az `OPTIMIZER_MODE` paraméter határozza meg, amelynek hatását felülírhatjuk egy konkrét lekérdezésben *tanácsok* (hints) segítségével. A tanács egy megjegyzésben elhelyezett optimalizálási utasítás egy lekérdezés belsejében.

A lekérdezésoptimalizáló által használt statisztikák az adatszótárban tárolódnak. Gyűjthetünk pontos vagy becsült statisztikákat az egyes sémaobjektumok fizikai tárolási jellemzőiről és az adatok eloszlásáról. Azon oszlopok esetén, amelyekben az ismétlődő értékek száma nagy változatosságot mutat, célszerű hisztogramokat gyűjteni. Az Oracle egyenlő szélességű és egyenlő magasságú hisztogramokat kezel. Ha az adott oszlopban a különböző értékek száma kisebb vagy egyenlő a hisztogrambeli csoportok számánál (amit a felhasználó ad meg), akkor egyenlő szélességű, különben egyenlő magassságú hisztogram jön létre.

Ezen statisztikák felhasználásával a lekérdezésoptimalizáló nagyobb pontossággal tudja kiszámolni az egyes tervek költségét, és így nagyobb biztonsággal találja meg a legkisebb költséggel rendelkező végrehajtási tervet. Ha nem áll rendelkezésre statisztika, az optimalizáló dinamikus mintavételezést végez az `OPTIMIZER_DYNAMIC_SAMPLING` paraméter értéke alapján, amely hosszabb feldolgozási időhöz vezet.

A lekérdezésoptimalizáló három lépésben határozza meg a végrehajtási tervet:

1. Előállítja potenciális tervek egy halmazát az elérhető hozzáférési utak és a tanácsok alapján.
2. Megbecsüli minden egyes tervnek a költségét az adatszótárban tárolt statisztikák alapján. A költség arányos a lekérdezés végrehajtásához szükséges erőforrások (lemez I/O, CPU, memória) várható mennyiségével az adott terv esetén. A magasabb költségű szekvenciális terveket több időbe kerül végrehajtani, mint az alacsonyabb költségűeket. Párhuzamos tervek esetén viszont a felhasznált erőforrások mennyisége nincs követlen kapcsolatban a végrehajtási idővel.
3. Összehasonlítja a kapott költségeket, és kiválasztja a legkisebb költséggel rendelkező tervet.

Az Oracle a 10g verziótól kezdve nem használja a szabály alapú optimalizálást, kizárólag a költség alapút.